# Example-Based Urban Modeling

Dissertation

zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

***Dipl.-Inf. (FH), Stefan Michael Hartmann, M.Sc***
aus Lichtenfels

Bonn, Januar 2018

Angefertigt am Institut für Informatik II mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

Dekan: Prof. Dr. Johannes Beck

1. Referent: Prof. Dr. Reinhard Klein
2. Referent: Prof. Dr. Andreas Kolb

Tag der mündlichen Prüfung: 20.07.2018
Erscheinungsjahr 2018 Diese Dissertation ist auf dem Hochschulschriftenserver der ULB Bonn
http://hss.ulb.uni-bonn.de/diss_online
elektronisch publiziert.

# Contents

**Disclaimer** v

**Zusammenfassung** vii

**Abstract** ix

**Acknowledgements** x

**1 Introduction** 1
1.1 Motivation . . . . . 1
1.2 Why Example-Based Modeling? . . . . . 2
1.2.1 Types of Content Generators . . . . . 2
1.2.2 The Example-Based Modeling Pipeline . . . . . 3
1.2.3 Strengths of Example-Based Modeling . . . . . 5
1.3 Why Example-Based Urban Modeling? . . . . . 6
1.4 Structure of the Thesis . . . . . 7
1.5 Contributions . . . . . 9
1.6 Publications . . . . . 10

**2 Review of Urban Content Generation Algorithms** 13
2.1 Techniques for Road Network Synthesis . . . . . 13
2.1.1 Road Network Growing Schemes . . . . . 14
2.1.2 Example-Based Road Network Generators . . . . . 14
2.1.3 Simulation-Based and Time-Dependent Approaches . . . 16
2.1.4 Countryside Road Generation . . . . . 17
2.1.5 Optimization-Based Approaches . . . . . 18
2.1.6 Controllable Road Network Generators . . . . . 19
2.2 Techniques for City Blocks Layouts Synthesis . . . . . 19
2.2.1 City Block Layouts using Subdivision Schemes . . . . . 20
2.2.2 Generation of General Discrete Element Layouts . . . . . 21

2.3 Techniques for Sequence Generation and Building Synthesis . . . 22
2.3.1 Fixed-Sized Sequences . . . . . . . . . . . . . . . . . . . . 22
2.3.2 Variable-Length Sequences . . . . . . . . . . . . . . . . . . 23
2.3.3 Resource Constrained Shortest Path . . . . . . . . . . . . 24
2.3.4 Synthesis of Buildings . . . . . . . . . . . . . . . . . . . . 25

**3 Efficient Optimization with Multiple Resource Constraints 29**
3.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
3.2 Resource-Constrained Sequences . . . . . . . . . . . . . . . . . . 30
3.3 Efficient Resource-Constrained Sequence Optimization . . . . . . 32
3.3.1 Single Resource-Constrained Optimization . . . . . . . . 32
3.3.2 Extension to Multiple Resource Constraints . . . . . . . . 35
3.3.3 Interval Constraints . . . . . . . . . . . . . . . . . . . . . 38
3.3.4 Extension to Higher Order Sequences . . . . . . . . . . . 38
3.4 Complexity Analysis . . . . . . . . . . . . . . . . . . . . . . . . 42
3.5 Comparison and Analysis . . . . . . . . . . . . . . . . . . . . . . 42

**4 Example-based Road Network Generation 45**
4.1 Hierarchical Road Network Generation . . . . . . . . . . . . . . 45
4.1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . 45
4.1.2 Hierarchical Fragment Construction . . . . . . . . . . . . 48
4.1.3 Hierarchical Synthesis of Street Networks . . . . . . . . . 50
4.1.4 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
4.1.5 Analysis and Comparison . . . . . . . . . . . . . . . . . . 59
4.1.6 Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . 60
4.2 Road Network Generation with GANs . . . . . . . . . . . . . . . 62
4.2.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . 62
4.2.2 Method Overview . . . . . . . . . . . . . . . . . . . . . . 66
4.2.3 Review of Generative Adversarial Networks . . . . . . . . 66
4.2.4 Towards Neurally Guided Road Network Synthesis . . . . 68
4.2.5 Case Study: Road Network Types . . . . . . . . . . . . . 72
4.2.6 Implementation Details . . . . . . . . . . . . . . . . . . . 80
4.2.7 Analysis and Comparison . . . . . . . . . . . . . . . . . . 81
4.2.8 Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . 82

**5 Example-Based Cityblock Layout Synthesis 85**
5.1 Example-Based Cityblock Layout Transfer . . . . . . . . . . . . 85
5.1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . 85
5.1.2 Problem Definition and Overview . . . . . . . . . . . . . 86
5.1.3 City Block Layout Transfer . . . . . . . . . . . . . . . . . 88
5.1.4 City Block Layout Refinement . . . . . . . . . . . . . . . 89

5.1.5 Example-Based 3D Building Placement . . . . . . . . . . 91
5.1.6 Synthesized Cityblock Layouts . . . . . . . . . . . . . . . 93
5.1.7 Synthesized Building Layouts . . . . . . . . . . . . . . . . 102
5.1.8 Analysis and Comparison . . . . . . . . . . . . . . . . . . 102
5.1.9 Limitations . . . . . . . . . . . . . . . . . . . . . . . . . 104
5.2 Guided Re-Synthesis of Discrete Element Arrangements . . . . . 107
5.2.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . 107
5.2.2 Re-synthesis Domain and Element Arrangements . . . . . 108
5.2.3 Efficient Re-synthesis of Element Arrangements . . . . . 111
5.2.4 Application: City Block Re-Synthesis . . . . . . . . . . . 116
5.2.5 Analysis and Comparison . . . . . . . . . . . . . . . . . . 120
5.2.6 Limitations . . . . . . . . . . . . . . . . . . . . . . . . . 121

**6 Example-based Building Synthesis 125**
6.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 125
6.2 Building Database and Annotations . . . . . . . . . . . . . . . . 126
6.3 Case Study: Resource Constraint Building Design . . . . . . . . . 128
6.4 Performance Evaluation . . . . . . . . . . . . . . . . . . . . . . 134
6.5 Analysis and Comparison . . . . . . . . . . . . . . . . . . . . . 135
6.6 Discussion and Future Work . . . . . . . . . . . . . . . . . . . . 135

**7 Summary 139**
7.1 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139
7.2 Example-Based Urban Modeling . . . . . . . . . . . . . . . . . . 142
7.3 Technical Future Directions . . . . . . . . . . . . . . . . . . . . 143
7.3.1 Road Network Synthesizers . . . . . . . . . . . . . . . . 143
7.3.2 Cityblock Layouts Synthesizers . . . . . . . . . . . . . . 144
7.3.3 Constrained Building Synthesizers . . . . . . . . . . . . . 145
7.4 Future Trends for Example-Based Modeling . . . . . . . . . . . . 146
7.4.1 Generation of Urban Structures . . . . . . . . . . . . . . 146
7.4.2 Content Storage and Compression . . . . . . . . . . . . . 147
7.4.3 Level of Detail . . . . . . . . . . . . . . . . . . . . . . . 147
7.4.4 Neurally Guided Content Generators . . . . . . . . . . . . 147

# Disclaimer

The content of this dissertation is based on three first author publications that have already been presented at different conferences with a focus on computer graphics.
The paper *StreetGAN: Towards Road Network Synthesis with Generative Adversarial Networks* [HWWK17] was published at the 'International Conference on Computer Graphics' (WSCG 2017). The paper presents a novel example-based approach for the generation of road networks using a recently introduced deep learning technique. I was responsible for the realization of the essential parts of the paper, including implementation, evaluation, and writing. My co-authors were responsible for proof-reading and technical discussions.
The paper *Content-Aware Re-targeting of Discrete Element Layouts* [HKK15] was published and presented at the 'International Conference on Computer Graphics' (WSCG 2015). In this publication, a re-synthesis scheme for 2D layouts composed of discrete elements is proposed. The approach is evaluated and exemplified by transferring a multitude of real-world city block layouts to virtual city blocks. I realized the prototypical implementation, evaluated the technique on an extensive portfolio of challenging real-world city blocks and did the write up of the paper. The co-authors were responsible for proof-reading and technical discussions.
The paper *Efficient Multi-Constrained Optimization for Example-Based Synthesis* [HTK+15] was published in the journal 'The Visual Computer' and was presented at the conference 'Computer Graphics International 2015'. It presents a novel optimization algorithm for example-based synthesis that allows the recombination of example parts such that multiple soft- and hard constraints are satisfied. The approach is evaluated for the concrete application of building synthesis using a database of buildings parts. I realized the essential components of the implementation and also produced the results that illustrate the different possibilities of the approach. My co-author Elena Trunz helped with technical discussions about the optimization technique and the preparation of the example data that was used to generate the buildings used for the evaluation. The other co-authors helped proof-reading the written parts of the paper and always had an open ear for discussions.

The dissertation contains a much more detailed analysis of the related work and includes an in-depth discussion of the published approaches. In addition, it adds additional chapters that present algorithms, yet unpublished. These include a hierarchical road network synthesis algorithm, a technique for the synthesis of building footprint layouts reusing the existing footprint layouts from real-world city blocks. To increase the geometric detail of a synthesized urban area a technique for the example-based placement of 3D building models reusing existing 3D building models is proposed. Finally, the last chapter of the thesis contains an entirely new view on future directions for the different technical parts and provides an extended, in-depth discussion of possible research directions in the field of example-based urban modeling. The work at hand contains text parts of the three mentioned papers that were transferred to this thesis without further modification. To make these copied sentences recognizable among the newly written parts they are highlighted in gray.

# Zusammenfassung

Um die Erzeugung virtueller Welten und insbesondere städtischer Umgebungen einem breiten Spektrum von Nichtexperten zugänglich zu machen, stellt diese Arbeit neue Methoden und Algorithmen zur automatischen beispielbasierten Erzeugung virtueller städtischer und urbaner Umgebungen vor. Beispielbasierte Generierung bedeutet in diesem Kontext, dass neue virtuelle Inhalte durch die Wiederverwendung von digitalisierten Geodaten, die als Beispiele oder Schablonen dienen, von Computerprogrammen automatisch erzeugt werden. Die notwendigen Geodaten, d.h. Straßennetze, Geländetopographie, Gebäudegrundrissdaten von Katasterämtern und detailliert modellierte 3D Modelle aus Gebäudedatenbanken, sind vermehrt über Internetdienste abrufbar und somit öffentlich zugänglich. Um diese Daten effizient und automatisch zu neuen virtuellen Straßennetzen, Stadtblöcken und individuellen Gebäuden zu rekombinieren ist die Entwicklung von neuen Methoden und Algorithmen notwendig. Diese kapseln Expertenwissen und erlauben es durch wenige Benutzereingaben, die Bestandteile virtueller Welten kontrolliert zu generieren. Der Fokus in dieser Arbeit liegt auf der automatischen Generierung von drei essentiellen Bestandteilen virtueller Städte: Straßennetze, Grundrissanordnungen in Stadtblöcken und individuelle Gebäude.

Zunächst wird in einem theoretischen Kapitel ein Optimierungsverfahren vorstellt, welches es ermöglicht, bespielbasierte Synthese als kürzeste Wege Problem unter der Einhaltung von Nebenbedingungen zu formulieren. Die Verwendung einer Hilfsdatenstruktur in Form eines gerichteten Graphen ermöglicht es Lösungen, welche die Nebenbedingungen nicht erfüllen, effizient zu verwerfen und somit von der finalen Pfadsuche auszuschließen. Hieraus ergibt sich ein verbessertes Laufzeitverhalten, wenn man die neue Methode gegen existierende Verfahren vergleicht. Die Basis einer stätischen Umgebung bilden Straßennetzwerke. Um virtuelle Straßen zu generieren werden in dieser Arbeit zwei neue Verfahren vorgestellt. Der erste Algorithmus extrahiert basierend auf einer hierarchischen Kategorisierung der Straßen in einem realen Straßennetzwerk Grundbausteine, die als Schablonen dienen. Diese kapseln Straßenmuster und Topographie auf unterschiedlichen Skalen. Ausgehend von einem Straßenskelett und

optionaler Geländetopographie, vorgegeben durch einen Benutzer, werden die Schablonen rekursiv rekombiniert, wobei die beste Passform über einen Formdeskriptor, erweitert um Topographieinformation, ermittelt wird. Auf diese Weise bildet sich so über mehrere Skalen wiederholt, ein neues Straßennetz. Auftretende Form- und Topographiedifferenzen werden durch Interpolation ausgeglichen. Das zweite Verfahren verwendet Erzeuger-Gegner Netze zur Generierung von Straßennetzen, eine kürzlich eingeführte Technik, die auf neuronalen Netzen basiert. Das in Vektordaten vorliegende Straßennetz wird als Binärbild kodiert, wobei Pixelintensitäten die Existenz sowie die Abwesenheit von Straßen kodieren. Das Erzeuger-Gegner Netz (GAN) wird anhand von Bildausschnitten fixer Größe trainiert, um die „datengenerierende“Verteilung zu lernen. Mit dem Erzeuger generierte Straßennetze spiegeln nicht nur visuelle Eigenschaften, sondern auch statistische Eigenschaften des Straßengraphs, wie Blockgröße oder Länge-Breite Verhältnis, wieder. Um die Stadtblöcke eines Straßennetzwerkes mit Gebäudegrundrissen zu füllen werden in dieser Arbeit zwei neue Techniken vorgestellt. Das erste Verfahren verwendet reale Straßenblöcke und überträgt die darin enthaltenen Gebäudegrundrissanordnungen in die virtuelle Welt, indem reale und virtuelle Stadtblöcke anhand ihrer Form verglichen und ausgerichtet werden. Darin enthaltene Grundrisse können anschließend in den Zielblock kopiert werden. Zur Evaluierung des Verfahrens werden eine Menge von Stadtblocklayouts synthetisiert und anschließend beispielbasiert mit 3D Gebäuden angereichert. Die zweite Technik generiert Gebäudegrundrisslayouts, wobei der Beispiel- und der Zielstadtblock in ihrer Form stark unterschiedlich sind. Durch das Konzept einer Führungskarte, d.h. einer Hilfsstruktur, die das gegebene Layout verzerrt in den Zielstadtblock abbildet, wird gezeigt wie sich die Grundrissanordnungen in einen formähnlichen aber nicht passgenauen Stadtblock übertragen lassen und somit den ursprünglichen Anordnungsstil erhalten. Die Synthese individueller Gebäude wird auf Basis des anfangs vorgestellten Optimierungsverfahrens als die Suche nach einem Resourcen-beschränkten kürzesten Pfad durch einen Graphen formuliert, wobei Knoten Gebäudebauteile detailliert modellierter Gebäude, die in einer Datenbank gespeichert sind, repräsentieren. Eine Evaluation des Verfahrens erfolgt durch die Synthese komplexer Gebäudestrukturen, welche mehrere lokale sowie globale Bedingungen erfüllen. Abschließend wird gezeigt, wie Gebäude von der Größe eines Stadtblocks generiert werden und anschließend zum Auffüllen leerer Stadtblöcke verwendet werden können. Die in dieser Arbeit vorgestellten Methoden haben das Ziel, möglichst wenige Benutzereingaben zu benötigen. Meist reicht eine abstrakte Zeichnung und die Vorgabe von Nebenbedingungen aus, um plausible Ergebnisse zu generieren. Manueller Aufwand besteht nur beim Aufbau von Datenbanken für Gebäudebauteile oder dem Herunterladen von Geodaten aus dem Internet.

# Abstract

The manual modeling of virtual cities or suburban regions is an extremely time-consuming task, which expects expert knowledge of different fields. Existing modeling tool-sets have a steep learning curve and may need special education skills to work with them productively. Existing automatic methods rely on rule sets and grammars to generate urban structures; however, their expressiveness is limited by the rule-sets. Expert skills are necessary to typeset rule sets successfully and, in many cases, new rule-sets need to be defined for every new building style or street network style. To enable non-expert users, the possibility to construct urban structures for individual experiments, this work proposes a portfolio of novel example-based synthesis algorithms and applications for the controlled generation of virtual urban environments. The notion example-based denotes here that new virtual urban environments are created by computer programs that re-use existing digitized real-world data serving as templates. The data, i.e., street networks, topography, layouts of building footprints, or even 3D building models, necessary to realize the envisioned task is already publicly available via online services. To enable the reuse of existing urban datasets, novel algorithms need to be developed encapsulating expert knowledge and thus allow the controlled generation of virtual urban structures from sparse user input. The focus of this work is the automatic generation of three fundamental structures that are common in urban environments: road networks, city block, and individual buildings.

In order to achieve this goal, the thesis proposes a portfolio of algorithms that are briefly summarized next. In a theoretical chapter, we propose a general optimization technique that allows formulating example-based synthesis as a general resource-constrained *k*-shortest path (RCKSP) problem. From an abstract problem specification and a database of exemplars carrying resource attributes, we construct an intermediate graph and employ a path-search optimization technique. This allows determining either the best or the *k*-best solutions. The resulting algorithm has a reduced complexity for the single constraint case when compared to other graph search-based techniques. For the generation of road networks, two different techniques are proposed. The first algorithm synthesizes a novel road net-

work from user input, i.e., a desired arterial street skeleton, topography map, and a collection of hierarchical fragments extracted from real-world road networks. The algorithm recursively constructs a novel road network reusing these fragments. Candidate fragments are inserted into the current state of the road network, while shape differences will be compensated by warping. The second algorithm synthesizes road networks using generative adversarial networks (GANs), a recently introduced deep learning technique. A pre- and postprocessing pipeline allows using GANs for the generation of road networks. An in-depth evaluation shows that GANs faithfully learn the road structure present in the example network and that graph measures such as area, aspect ratio, and compactness, are maintained within the virtual road networks. To fill empty city blocks in road networks we propose two novel techniques. The first algorithm re-uses real-world city blocks and synthesizes building footprint layouts into empty city blocks by retrieving viable candidate blocks from a database. We evaluate the algorithm and synthesize a multitude of city block layouts reusing real-world building footprint arrangements from European and US-cities. In addition, we increase the realism of the synthesized layouts by performing example-based placement of 3D building models. This technique is evaluated by placing buildings onto challenging footprint layouts using different example building databases. The second algorithm computes a city block layout, resembling the style of a real-world city block. The original footprint layout is deformed to construct a *guidance map*, i.e., the original layout is transferred to a target city block using warping. This guidance map and the original footprints are used by an optimization technique that computes a novel footprint layout along the city block edges. We perform a detailed evaluation and show that using the guidance map allows transferring of the original layout, locally as well as globally, even when the source and target shapes drastically differ. To synthesize individual buildings, we use the general optimization technique described first and formulate the building generation process as a resource-constrained optimization problem. From an input database of annotated building parts, an abstract description of the building shape, and the specification of resource constraints such as length, area, or a number of architectural elements, a novel building is synthesized. We evaluate the technique by synthesizing a multitude of challenging buildings fulfilling several global and local resource constraints. Finally, we show how this technique can even be used to synthesize buildings having the shape of city blocks and might also be used to fill empty city blocks in virtual street networks. All algorithms presented in this work were developed to work with a small amount of user input. In most cases, simple sketches and the definition of constraints are enough to produce plausible results. Manual work is necessary to set up the building part databases and to download example data from mapping services available on the Internet.

# Acknowledgements

At the very first, I would like to thank Prof. Dr. Reinhard Klein who supervised me during the years in his computer science department. His comments and discussions about different ideas and techniques to solve problems really advanced my way of working on different research topics.
I am also very grateful to Prof. Dr. Andreas Kolb who kindly agreed to serve as an external reviewer.
Additionally, I would like to thank my co-authors, Elena Trunz, Michael Weinmann, Björn Krüger, Matthias B. Hullin, Michael Weinmann and Raoul Wessel.
In addition, I would like to thank the AiF Project GmbH for funding the project AtEgosim under the grant KF2179204LF2 and the BMBF for funding the project USUSim under the grant 01ISO9043B.
Last but not least I would like to thank my wife Franziska and my son Arthur for supporting me in the previous years - thank you.

# CHAPTER 1

# Introduction

## 1.1 Motivation

Today's simulation environments and video games represent huge virtual worlds that are composed of rich and high-quality virtual content. Even digital movie productions cannot possibly be imagined without highly detailed virtual fantasy worlds. These worlds are represented as 2D or 3D scenes of varying size and consist of a significant amount of digital assets. To achieve a convincing perception for the observer, a huge amount of varying content having different levels of detail is necessary. In the early days of digital content production, the virtual assets representing different types of content were modeled from scratch by a large pool of 3D artists and then composed to larger scenes. A lack of easy to use design tools makes the production of digital content an extremely arduous and cumbersome task. Sometimes, even specialized technical skills and extensive training might be necessary to use well-established design tools such as Maya©, 3DSMax©, or Modo©, in a time efficient and productive way.

Instead of modeling content from scratch, automatic data acquisition based on different capturing techniques such as 3D scanning or photography might be used. While the quality of digitized real-world artifacts might be higher, the problem of efficient content creation is just shifted. Many cases disallow the direct use of captured content as artifacts might be present, thus requiring additional time-consuming human assisted post-processing. It should not be forgotten that the time for capturing content also includes the time to transfer the capturing device to a specific location. In addition, capturing content typically produces large amounts of data that need to be stored somewhere, and therefore only a few objects might be captured at a given point in time.

One possible approach to reduce the time and cost of modeling novel content is

using algorithms as workhorses to support 3D artists. Such algorithmic content generators can either produce virtual assets in a fully automatic fashion or might be interactively guided by a user. For the efficient design of virtual content generators, several questions arise: How can an algorithm generate content that is perceived as plausible and mimics the style found in real-world examples? What type of content can be generated by what sort of algorithm? What input data, parameters, or user input is necessary to produce content that even contains variations efficiently? What type of algorithmic abstraction is required to make a single content generator capable of synthesizing plausible results for different types of content?
To answer these questions, the efficient design and editing of content for digital media in general, and computer graphics, in particular, continues to be one of the most prolific subjects of investigation within the computer graphics research community. Intensive efforts to close the productivity gap of typical 2D and 3D modeling work-flows have given rise to novel models and user interfaces that make the design and editing process of digital content more intuitive and efficient. Approaches that were developed within the computer graphics community in recent years range from powerful algorithms for the synthesis and re-targeting of animation sequences, editing and deformation of shapes and shape collections, parametric and data-driven texture synthesis, and even geometry synthesis and quilting.

## 1.2 Why Example-Based Modeling?

The generation of novel content that is perceived as plausible and even mimics the style found in real-world examples is a vibrant research topic within the computer graphics community. Apart from the questions, *What types of content can be generated?* and *What data might serve as input to the algorithm?*, an essential aspect in the field of algorithmic content generators is the type of algorithm that produces novel objects. When reviewing, the area of automatic and semi-automatic content generation algorithms, typically three types of content generators arise: *procedural content generators and example-based content generators.*

### 1.2.1 Types of Content Generators

**Procedural Content Generators:** Procedural approaches are steered by a set of deterministic production rules in combination with a set of parameters. The rules are evaluated, and a deviation structure is constructed that is used to produce certain types of content. As only rules and parameters are necessary, this representation for virtual content is very compact. Varying content is achieved

by randomness in the deviation of specific rules. However, even with carefully designed statistical variability, the output generated by this class of methods is limited by the expressiveness of the rule-set. In many cases, the rules are defined manually; however, the manual design of the rules is still an an incredibly tedious task. The reader is referred to recent literature for an in-depth discussion of procedural algorithms [SKUF15, ADBW16].

**Example-Based Content Generators** Example-based approaches are fed with a single example or a set of examples stored in a database. The exemplars might be taken from existing data sets or might be captured for a specific application purpose. Sometimes, the input examples might be further split into sub-parts in a pre-processing step. These sub-parts form the basic building blocks from which novel content is built. A custom-tailored algorithm utilizes these buildings blocks, determines different parts that fit together and then copies, re-shuffles, re-combines, or bends them produces novels objects.
When the content generator only re-uses the original parts, by just copying them to form a possibly larger output the content generator is said to be purely *data-driven*. When the input is analyzed in a first step, statistical information is extracted, a complex model is learned, or the content is deformed or bend, then the more general term *example-based content generator* might be utilized.

### 1.2.2 The Example-Based Modeling Pipeline

Example-based content generators were introduced to the computer graphics community to reduce the productivity gap further and lessen the obstacles for fast and intuitive content generation. Early example-based modeling pipelines were documented in Cohen [M00] or Funkhouser et al. [FKS$^+$04]. In these early pipelines, four basic building blocks can be identified that are part of each example-based modeling pipeline (see Figure 1.1). Even in more recent publications about example-based modeling, these building blocks can still be found. In the following, each of the major blocks and its primary task is briefly described.

**Data Acquisition** In a very first step, the examples serving as input to an example-based algorithm need to be acquired. These might be images, 3D models, videos or vector graphics. With the emergence of web-services, large databases that already contain plenty of data can be tapped for designing and implementing example-based modeling pipelines. Examples for such databases that contain plenty of existing content are *Textures.com* [Tex17] for texture resources, *Turbosquid* [Tur17] and *Trimble Warehouse3D* [Tri17] for general 3D models or *Earth Explorer* [Ear17] for satellite images, roads networks, or topography data.

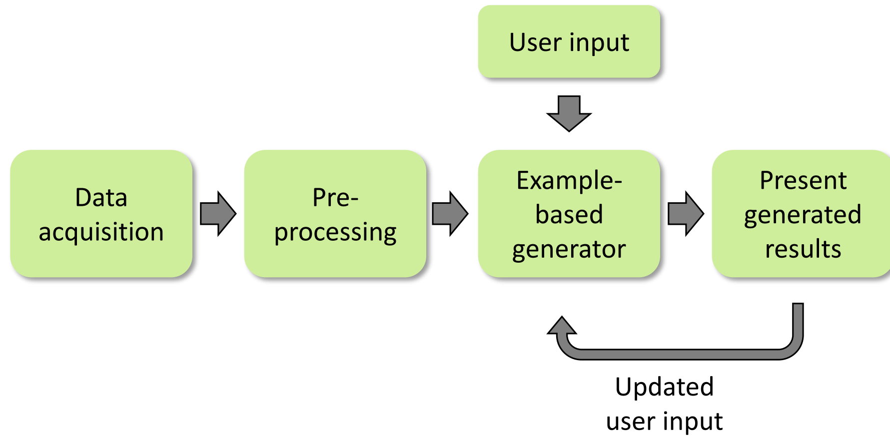



Figure 1.1: Overview of the four essential steps that are present in example-based modeling pipelines.

In cases where no specific data is available or re-usable, it might be acquired for a specific application to use the example-based modeling metaphor. Concrete examples can be found in the works of, Pauly et al. [PMG+05] for 3d scan completion, Lerner et al. [LCL07] for moving crowd video data, Xie et al. [XYS+16] for example-based tree modeling, or Ruiters et al. [RSK13] for interpolation of bidirectional texture functions.

**Data Preprocessing** Although plenty of data that might serve as input for example-based modeling techniques exists, the data gathered from these databases is typically not directly usable as input to the content generator. In a first step, the data might be filtered to only extract the necessary content parts, or it might be transformed into a different representation that simplifies further processing. In a second step, it might be further split or decomposed into meaningful parts that are enriched with statistics or semantics. The result is then typically stored in a database and serves as input to concrete synthesis algorithms.

**Content Generator Algorithm** The core of the modeling pipeline is in many cases a custom-tailored algorithm that re-uses the exemplars to generate novel content. In many cases specifications provided by a user serve as additional input to the content generator. These might come in the form of sketches, scribbles, preferences encoded as weights, or semantic maps and are generally used to control and constrain the output produced by the content generator. Algorithms that generate content are typically designed to achieve one or multiple intended design

goals. The core task of the content generators is to perform matching of exemplars against each other and/or the provided user input and to determine a set of candidate exemplars for further processing. The outcome from the candidate selection is used to evaluate one or multiple objective functions and to perform constraint satisfaction checks. Finally, the examples are combined by applying different techniques such as copying, warping, or mixing techniques. During this step, additional tests are performed to ensure that the previously defined constraints are not violated. Depending on the application domain, the algorithm might produce a single optimal solution or a set of solutions.

**Presentation of the Results** After the previous steps have produced the content, the best solution or multiple most probable solutions are presented to a user. He or she then selects the result that closest fits their demands. In any case that she might not be satisfied with the results, the provided user input might be updated, or the example database might be manipulated or exchanged, and the algorithm is run once again. According to Fisher et al. [FRS+12] a single solution or multiple solutions should be presented, and they state that a suitable heuristic might be: every third solution should be a 'good' one.

### 1.2.3 Strengths of Example-Based Modeling

The example-based modeling metaphor has several strengths that make it a good choice for specific target applications. The most important requirement to successfully apply this methodology is the presence of available data that might serve as input for the content generators. When existing data repositories and publications are reviewed, there is indeed high-quality content available for nearly any category of object types that are commonly used for the design of virtual worlds. A few resources of exemplars that have been used in publications and that were made publicly available are in particular motion segments [MRC+07], building and room layout [SYZ+17, BLS16], textures [Tex17, CMK+14, DRV14], architectural textures [DRSV13].
The content generation algorithm is typically designed to reduce the workload of a possible user drastically. The tedious work that is necessary to recombine or reshuffle the examples manually is delegated to an algorithm. Thus, the algorithm needs to encapsulate the 'knowledge' or 'best-practices' of a domain specific expert, that knows how to recombine existing content parts. This is extremely useful, as it suddenly allows non-expert users to create novel digital content effectively. On the other hand, experts can create novel content more productive and time efficient. In addition, the synthesized content might support the creativity of a user as it effectively allows to browse the solution space and to inspect or to analyze how different exemplars might be combined in different ways.

The exemplars that are fed to the content generator are typically extracted from real data and stored in a database. For different target applications, the existing content may be available in a similar form. For example, music and animation sequences are time-series of varying content types. Textures are typically represented as 2D image patches, and 3D objects might be represented as volume elements. A transformation of the exemplars into a more abstract representation allows designing content generation algorithms that can produce novel content for utterly different target applications.
One benefit of the example-based modeling metaphor is that users can produce rich and detailed content with only a small amount of input. In many cases, the input needs to be specified using just a few simple operations such as sketching, scribbling, and the selection of a database that stores the exemplars. As the user interface to interact with the content generator is typically clearly arranged and intuitively designed, the tools are easy to learn. Thus, a short introduction of the overall interface and an explanation of the possible operations are enough to allow non-expert users to efficiently design, edit, and re-design novel content. For these reasons, the example-based methodology significantly reduces the user interaction with the content generation system to three steps: (1) **acquisition** of data from which the exemplars are derived, (2) provide **input** and constraints, (3) **selection** of candidates among the synthesized results.

## 1.3 Why Example-Based Urban Modeling?

Especially with the emergence of a large variety of publicly available content repositories for geographical data, that are maintained by large communities, modeling by example has also been exploited for the synthesis of urban environments. With access to mapping services such as OpenStreetMap [Ope17a] or EarthExplorer [Ear17], planet-scale real-world data has become available. It contains examples ranging from satellite imagery, terrain data, road networks data and building information of near any town or settlement of the world. Even land surveying offices offer publicly available multi-purpose cadastre data such as detailed building footprints and parcels layouts. The existence of such publicly available data allows creating virtual replicas of the real world and even the design and generation of novel virtual urban environments. This allows a variety of different applications such as driver training, the planning of urban neighborhoods, or virtual architectural design.
Reusing the real-world data also allows the creation of novel virtual worlds that demand specific requirements. These requirements are typically necessary to perform human behavioral studies, a particular type of experiment where the reaction of an individual user is evaluated during an individual experiment. Such scenarios

were the demand of our project partner, during the first years of my thesis. In more detail, the partner had a concrete need for tools and algorithms that can be used for the intuitive creation of virtual urban environments that should be used for the conduction of individual virtual driving experiments. The virtual urban spaces, where the experiment will take place should be created by psychologists who plan and conduct the experiments. Thus, the potential users of the modeling tools are users who are non-experts in the field of urban modeling. In addition, these people were not familiar with standard 3D modeling tools such as Blender, Maya or 3D Studio Max and they also had no experience with procedural urban modeling systems such as CityEngine [Esr17].
Such a situation calls for urban modeling tools that have a flat learning curve, are easy to use, and require only a small amount of user input. The virtual worlds to be created should mimic the real world to provide a harmonious and comfortable environment for a specific test-person and might be edited and updated until they fit the psychologists' needs. In combination with all the necessary data available via online services such as *OpenStreetMap* [Ope17a], EarthExplorer [Ear17] and Trimble Warehouse3D [Tri17], the example-based modeling metaphor seems to be a promising choice, especially for the development of novel content generation algorithms, for the design of urban spaces. For these reasons, the primary goal of the work at hand is to provide a portfolio of algorithms to synthesize three types of structures that are necessary to model detailed virtual urban environments: *road networks*, *city block layouts* and *building layouts*.

## 1.4 Structure of the Thesis

In here, we briefly give an overview of the structure of the work at hand. The current introductory chapter is followed by an in-depth discussion of related work on the generation of different urban structures such as *road networks*, *city block layouts* and *buildings* and *building layouts*. In chapter 3, a general algorithm for resource-constrained optimization for example-based modeling is introduced. The algorithm is uncoupled from specific types of exemplars and allows to be used in different content generation tasks, such as human motion synthesis, sound synthesis, image synthesis, and building synthesis. By using an intermediate graph representation, the algorithm achieves superior performance to previous work, as only resources feasible solutions are traversed during the optimization. The applicability of the proposed algorithm is exemplified by the generation of buildings and city blocks that satisfy different sets of local and global constraints.
In chapter 4, we put the focus on the generation of road networks, the skeleton of nearly every urban layout. The content of chapter 4 consists of two different road network synthesis strategies. The first algorithm synthesizes a new road network

from a hierarchical set of fragments. The fragments are automatically extracted from publicly available real-world street data and can be recursively recombined taking terrain topography into account. In contrast to this algorithm, the second road network synthesis technique employs machine learning techniques for the generation of street networks. It is based on generative adversarial networks (GAN), a recently introduced deep learning technique. The structure of an example network is learned by first training the GAN on a raster representation of the example road network. Novel road networks can be synthesized by feeding values drawn from a simple distribution to the generative component of the GAN and extracting the road graph from the produced image and transforming it into a vector-based representation.

The space that is covered by a road network is divided into small areas enclosed by roads - the city blocks. For increasing the realism of a city layout, it is crucial to fill these city blocks with realistic building footprint arrangements. We focus on this problem in chapter 5 and present two different solutions that transfer the building footprint layout from real-world city blocks into the city blocks of the virtual street network. The first algorithm (see section 5.1) approaches the synthesis of building footprint layouts using the shape of the virtual city blocks and retrieves a viable candidate by comparing the shapes of the city blocks. After alignment of both city blocks, the building footprints from the retrieved candidate block can be copied into the target block. Potential overlaps with the street area and pavement are resolved using a constrained optimization technique. Additional details are added to the city block layout by example-based building placement using databases containing 3D building models. In here, a viable candidate building is retrieved from a database by comparing the shapes of the footprints. The second algorithm targets city blocks where the shape between the virtual and the real city blocks strongly differ. To properly transfer the real-world building footprint layout to the virtual world, we propose a re-synthesis scheme in section 5.2. Here, the original footprint layout is warped into the target city block. This deformed layout is used as a guidance map to compute a new building footprint layout reusing the original building footprints as building blocks.

In chapter 6 of this thesis, we focus on the synthesis of the smallest entities of an urban layout - the buildings. In contrast to chapter 5 where existing buildings were reused as a whole, we focus on the generation of new buildings using a database of building parts. The building parts carry resource attributes and can be recombined and reshuffled to form new buildings maintaining the style of an example building. We use the optimization technique proposed in chapter 3 to synthesize a multitude of individual buildings and city blocks that satisfy several local and global constraints. In addition, we show that the synthesized building blocks can be used to fill the empty city blocks within virtual and real-world street networks.

## 1.5 Contributions

The work at a hand proposes a portfolio of algorithms and applications to ease the controlled generation of urban structures, heavily re-using real-world exemplars. The algorithms range from the synthesis of macroscopic urban structures to the generation of microscopic urban structures as already briefly discussed in section 1.4. In particular, the contributions include:

- **Graph-Based Constrained Optimization:** We formulate example-based synthesis as a general resource-constrained *k*-shortest path (RCKSP) problem. An abstract problem specification and a database of exemplars carrying resource attributes serve as input. We approach the problem using a novel two-step approach. First, we construct an intermediate graph representation, containing only resource feasible solutions. Second, we employ a path-search optimization technique to determine either the best or the *k*-best solutions. An in-depth analysis shows that the proposed method has a reduced complexity for the single constraint case when compared to other graph search-based techniques.
- **Road Network Synthesizers:**
  - We propose a purely example-based approach for the generation of road networks from sparse user input, i.e., a sketch of a desired arterial street skeleton and a height map that represents the desired topography. We exploit the natural hierarchy of road networks to extract a set of hierarchical, closed fragments that form the central building blocks for the road network synthesizer. We present an algorithm that recursively constructs a new road network by retrieving viable fragments from a database and inserts them into the current state of the road network, while the difference in shape will be compensated by warping both the interior street topology and the topography of the best matching candidate fragment.
  - We propose a novel method for the synthesis of road networks using generative adversarial networks (GANs), a recently introduced deep learning technique. We develop a pre- and postprocessing pipeline to use GANs for the generation of road networks. We provide an in-depth evaluation that shows that GANs faithfully learn the road structure present in the example network and that the synthesized road networks even maintain graph measures such as area, aspect ratio, compactness when compared to those of the original network.
- **City Block Layout Synthesizers:**

  - We propose a layout transfer algorithm for filling real-world building footprint layouts into empty city blocks of real or virtual road networks. We evaluate the algorithm by synthesizing a multitude of city block layouts reusing real-world building footprint arrangements from European and US-cities. We increase the realism of the synthesized layout by performing example-based placement of 3d building models and evaluate this strategy on challenging footprint layouts with different example-building databases.
  - We develop a re-synthesis scheme for computing new city block layouts that resemble the style of a real-world city block. We propose to transfer the layout of a real-world city-block into a virtual city block using warping to construct a *guidance map*. We present an efficient labeling algorithm that uses the computed guidance map and the building footprints shapes from the real-world city block to synthesize a novel layout within the virtual city block. We perform an in-depth evaluation that shows, that the use of the guidance map allows transferring the style of the original layout locally as well as globally even when the source and target shapes drastically differ.

- **Example-based building synthesis:** We use the graph-based optimization framework to formulate the synthesis of buildings as a constrained optimization problem that uses the proposed resource constrained $k$-shortest path algorithm. From an input database of annotated building parts, a skeleton that describes the shape of the building, and the specification of resource constraints such as length, area, or a specific number of architectural elements, a novel building is synthesized using the mentioned *Graph-based Constrained Optimization* technique. We evaluate the method by synthesizing a multitude of challenging buildings fulfilling several global and local resource constraints and that even can be used to fill in empty city blocks in road networks.

## 1.6 Publications

As part of the work at hand, the following four papers were published at different computer graphics conferences. Please note, that the thesis only describes and discusses the content of the publications where the author of this thesis is also the first author of the individual publications.

1. Stefan Hartmann, Elena Trunz, Björn Krüger, Reinhard Klein, and Matthias B. Hullin. Efficient multi-constrained optimization for example-based synthe-

sis. *The Visual Computer / Proc. Computer Graphics International (CGI 2015)*, 31(6-8):893–904, June 2015

2. Stefan Hartmann, Björn Krüger, and Reinhard Klein. Content-aware retargeting of discrete element layouts. In *International Conference on Computer Graphics, Visualization and Computer Vision*, volume 23 of *WSCG proceedings*, pages 173–182, June 2015

3. Stefan Hartmann, Michael Weinmann, Raoul Wessel, and Reinhard Klein. Streetgan: Towards road network synthesis with generative adversarial networks. In *International Conference on Computer Graphics, Visualization and Computer Vision*, June 2017

4. Jonathan Klein, Stefan Hartmann, Michael Weinmann, and Dominik L. Michels. Multi-scale terrain texturing using generative adversarial networks. In *Image and Vision Computing New Zealand*, December 2017

# CHAPTER 2

# Review of Urban Content Generation Algorithms

In this chapter, we review literature that is closely related to example-based urban modeling. We focus on the review of three essential urban structures: street networks, city blocks, and building synthesis techniques. Additional readings in the context of urban modeling can be found in Vanegas et al. [VAW$^+$10] who focus on appearance of urban spaces, Wonka et al. about procedural urban modeling and simulation [WMWF07, WAMV11], Musialski et al. about urban reconstruction [MWA$^+$13], or Aliaga et al. [ADBW16] who provide an extensive tutorial on inverse procedural modeling techniques for urban structures.

## 2.1 Techniques for Road Network Synthesis

Road networks represent an essential part of an urban ecosystem and thus form an important component of an urban environment. For the design of virtual urban environments, it is crucial to have algorithms that are able to synthesize road networks that mimic the style found in real-world examples. Different requirements on the virtual road networks have spawned several individual solutions that might be used for the design of cities, villages or neighborhoods. Among them, one finds approaches that produce content that might be suited for game design or movie productions. However, there exist methods that try to advance the process of virtual urban planning and focus on the generation of high-quality road structure and parcel layouts. The goal of this section is to provide a brief overview of the various techniques that are closely related to the work at hand.

### 2.1.1 Road Network Growing Schemes

An early approach for procedural road network generation was introduced by Parish and Müller [PM01]. Their work utilizes *extended L-Systems* to procedurally grow road networks satisfying global goals and local constraints. The generation process is separately performed on two different street hierarchy levels, *major* and *minor* using different re-writing rules controlled by several rule parameters. In order to follow global goals, the placement of roads is constrained to regions of high population density. Major roads use rules to achieve a certain road network pattern type such as grid, radial, or irregular style. In contrast to global goals, local constraints adopt the rule parameters in order to avoid the placement of illegal road constellations such as steep road segments or roads touching the water. Furthermore, local constraints also provide a snapping mechanism to connect newly generated road segments to existing nearby located crossings or road segments. Kelly and McCabe [KM07] follow the road generation procedure of Parish and Müller and synthesize roads on two street levels, i.e. major and minor roads. However, in contrast to Parish and Müller they do not use a string based rewriting system, but let the user specify the high-level topology of the road network in a preprocessing step. The major road courses are then generated by sampling and plotting roads that smoothly adapt to the underlying terrain. Minor roads are then generated by first placing seed points along the 'edges' of the major road graph and second using a parallel growth algorithm inspired by L-systems.
While the previous approaches focused on the road network generation of cities or city-like structure, the automatic generation of sparse settlements or villages has been investigated by Emilien et al. [EBP$^+$12]. The roads of a village skeleton are constructed across the terrain utilizing an anisotropic shortest path algorithm that is guided by several user adjustable objective functions. In order to avoid the generation and placement of a huge amount of different nearby located road branches, the weights along existing road courses are re-weighted. This enables the shortest path algorithm to re-use existing road structures heavily and allows synthesizing plausible looking road courses found in villages.

### 2.1.2 Example-Based Road Network Generators

Sun et al. [SYBG02] propose to use templates for the design of novel road networks. They define templates as 'modules' that encapsulate an algorithm that produces an individual road network pattern. From a set of population centers that serve as sites for the computation of a Voronoi diagram, a road network can be constructed by interpreting the edges of the diagram as roads. Grid structured and radial structured road patterns are produced by encapsulating growing algorithms with specific rule sets steered by parameters into templates.

More recent approaches for road network generation make use of templates in order to synthesize road structures. However, compared to Sun et al. [SYBG02] the templates represent different sized patches of roads that are either extracted from a real-world road network or hand-crafted and enriched by manual annotations. Yang et al. [YWVW13] use hand-crafted templates to synthesize the high-quality road network structures found in suburban layouts that follow urban design guidelines. They propose a recursive subdivision scheme that splits an initial empty domain into smaller sub-domains using templates that represent virtual replicas of road patterns, commonly found in suburban regions. The basic idea of the algorithm is to match the outline of a template to the coarse shape of a closed empty region within a virtual road network. The candidate template is then iteratively deformed and reoriented to closely match the shape of the region that shall be split. When the initial deformation energy is too high, a fall-back subdivision is achieved by computing streamlines that are utilized to subdivide the region into different parts without using any templates.
Aliaga et el. [AVB08] follows a different line. In their work, they use crossings enriched with statistical information as templates. An existing road network and a set of arterial streets is used to compute a road hierarchy using stream ordering [Hol94] including loops. Each crossing within the original road network is annotated with incoming street levels and statistical information such as the distance between consecutive crossings, the angles between adjacent road segments, and curvature. By re-distributing the existing crossings in a new urban area, a random walk algorithm constructs the new road network topology starting from the highest hierarchy level. It utilizes the information stored along with the crossing in order to iteratively form the most likely connection to nearby located crossings. One disadvantage of using enriched crossings [AVB08] or crossing templates [YS12] as input for the road network generator is that essential road structures such as ramps, roundabouts, and plazas cannot be faithfully reproduced within the new road topology. This important issue was later addressed by Nishida et al. [NGDA16a] where templates represent patches of roads extracted from real-world road networks such as *OpenStreetMap* [Ope17a]. Road segments at the patch boundary represent connector segments that accept connections to other patches when they share a similar length. The user feeds the system with an initial seed point, that is used by the algorithm to perform example-based growth steps. This particularly means that the road network is expanded in a breadth-first manner utilizing a database of extracted example-patches. At each growth step, local constraints similar to Parish and Müller [PM01] are evaluated in order to either accept the network expansion or to perform a procedural fall-back similar to Aliaga et al. [AVB08].
In Yu and Steed [YS12] crossings with their adjacent road segments are used as templates. Their approach is a combination of the two algorithms presented

in [AVB08] and [PM01]. From an initial seed point, the road network is iteratively expanded. Their method searches the example network for a region that is topologically and geometrically similar to the neighborhood of the currently processed crossing with its adjacent road segments. The size of the neighborhood used typically depends on the size of the structures that should be captured within the example network. They expand the road topology after performing, local constraint checks similar to [PM01, KM07].
Emilien et al. [EVC$^+$15] learn distributions from small patches of generated content or manually modeled 3D content. The learned distributions are applied in a brushed-based modeling metaphor in order to steer the underlying procedural content generators that produce roads, foliage, trees, or buildings. Their road network generation component extends the one of [AVB08] by avoiding the cumbersome manual distribution of crossings. Instead, they learn a radial distribution function inspired by recent works on point distributions by Otztereli and Gross [OG12]. Connections between crossings are then achieved using the random walk method of [AVB08].

### 2.1.3 Simulation-Based and Time-Dependent Approaches

The approaches discussed so far were tailored to produce static road networks representing a fixed point in time. However, for urban planning or simulation games, these road network generators are not suited. For dynamic cites a different class of algorithms for geometric and behavioral modeling is necessary. These types of algorithms simulate the dynamic growth of one or multiple cities over time by taking urban attributes such as traffic, population, jobs, land use, or land value into account. In contrast to early approaches that simulate urban development [Wad02, WBN$^+$03] using drastically simplified models for simulation of urban spaces, more recent approaches integrate detailed geometric aspects into the simulation loops. In Weber et al. [WMWG09] a system for the temporal simulation of a 3D urban model is presented. Instead of performing the simulation on a regular grid (see Waddel [Wad02, WBN$^+$03]), the simulation takes exact geometries, such as street geometry, parcel boundaries, or building footprints into account. Their road network growth is inspired by Parish and Müller, but instead of using a rewriting system they stochastically choose high probability nodes located nearby growth centers. The street is grown according to a rule set and needs to adapt to the environment similar to the local constraints of Parish and Müller. The main difference is that the produced segments are just planned; however, their final position is already fixed. As common in geometric urban modeling, first major roads are grown, while minor roads are processed in a second step. The traffic is simulated by performing virtual trips between street segments. These depend on the population and the attractiveness at certain locations and include already

planned segments. When enough traffic demand for planned street segments is generated, they finally become active after a user-defined threshold is passed. In contrast to the common process of generating major roads or highways in a first step and minor roads in a second one, Benes et al. [BWK14] reverse the road generation process. From an initial major road network, minor roads are grown in the first step around identified city cores. Paths along major roads connecting these cores are utilized to perform a traffic simulation for pairs of neighboring cities to determine a potential increase in traffic and thus the demand for new major roads. These are generated utilizing special rules for major road growth that introduce novel road segments or transform existing minor segments into major ones.
Vanegas et al. [VABW09b] couples behavioral modeling and geometric modeling in an interactive design system. The behavioral simulation is inspired by Waddel [Wad02] who simulate urban development on spatial grids, where cells capture urban attributes such as land use, population, jobs, and accessibility. Their focus is on editing and designing a new urban model instead of simulating the growth over time [WMWG09, BWK14]. A user can change any geometric or behavioral attribute, while a dynamic system assures an equilibrium state, a state where the behavioral model matches the geometric model and vice versa. In their model streets are grown at three different levels: highways, arterial roads, and local roads. Intersection seeds are computed by identifying job and high population clusters. Streets are generated in a breadth-first manner, taking the underlying cell attributes into account while higher level roads, i.e., arterial and local roads are grown from a mixture of seed points computed by a clustering approach that is augmented with seed points placed on the streets of the parent street level. This ensures that higher level roads are always connected to already existing street structures.
Krecklau et al. [KMK12] use a data-driven approach for the interpolation of cities instead of a model-driven one [Wad02, VABW09b] that typically requires a huge amount of professional and expert knowledge. They rely on key points in the past, i.e., historic city maps that represent intermediate interpolation targets. An event system simulates the construction and destruction of specific urban objects, i.e. streets and buildings. The events are stored in a dependency graph, that reflects relations between different types of events. In a simulation loop, the events are categorized into different priorities and are subsequently executed. The system automatically detects event conflicts and asks the user to resolve them interactively.

### 2.1.4 Countryside Road Generation

Another important road structure type is cross-country roads. These roads form vessels across the country to enable transit connections between different urban re-

gions. The synthesis of such road structures is proposed by Galin et al. [GPMG10]. The authors focus on the generation of a path that connects two arbitrary selected points on a landscape. The underlying terrain space is discretized into cells, that are used to compute the shortest path between a user-specified start and endpoint. During the path computation, several cost functions are evaluated, that guide the construction of tunnels, bridges, and road segments across the terrain. To increase the variability in possible direction changes the path generator is allowed to skip neighboring cells and directly advance to other cells in a user-defined $n \times n$ neighborhood. While this approach is only able to connect a single start and destination point, Galin et al. [GPGB11] advance this approach to generate connection networks for different sized urban areas, i.e., cities, towns, and villages. These are connected using different types of roads, i.e., highways, major roads, and minor roads. An additional cost function penalizing or favoring different terrain regions guides the shortest path algorithm connecting the urban structures. In contrast to Emilien et al. [EBP+12] (see Section 2.1.1) they do not have a mechanism to allow road re-use and therefore propose to merge roads that are closely located next to each other.

### 2.1.5 Optimization-Based Approaches

In a different line of research, optimization techniques are used for the design of road structures. Such methods leverage the power of recent combinatorial optimization solvers to enable road network designs that fulfill different types of hard and soft constraints. The approach of Peng et al. [PYW14] proposes to compute a discrete tiling of arbitrarily shaped domains that are represented as a quadrilateral mesh. The tiling is composed of building blocks where each of them is itself formed from a set of quadrilaterals and can be imagined similar to *Tetris* blocks. Their outline edges can be interpreted in an application-specific way. For the synthesis of road networks, the outline of the tiles represents individual road segments. The authors formulate the tiling problem as an integer linear program (ILP), where the building blocks represent the variables of the optimization problem. Such a tiling problem can be solved using off-the-shelf solvers that perform well when the number of building blocks is small. One advantage of the ILP formulation is that different types of constraints such as the number of T-crossings or the occurrence of specific building blocks can be easily controlled. ILP-based formulations for the synthesis of functional networks were extended in the work of Peng et al. [PYB+16]. In their formulation, a network is now represented as a set of activated edges within the initial domain that is composed of quadrilateral elements. The vertices of the quadrilateral mesh might serve to model different types of constraints such as sink constraints or point-to-point constraints. The first constraint type allows forcing the network to start or end at a subset of vertices,

while the latter allows for producing connections between a subset of vertices.

### 2.1.6 Controllable Road Network Generators

Apart from synthesizing novel content, editing and updating the generated content has been established as an essential research direction within the computer graphics community. Controlling the procedural generation of road networks was tackled by Chen et al. [CEW+08], who let the user define and edit two-dimensional tensor fields. The user has access to different types of tensor fields such as radial or grid ones. The procedural road network algorithm is aware of the tensor field and traces hyper streamlines along the major and minor eigen-directions of the tensor field. They utilize a modified Runge-Kutta integration scheme and an interleaved tracing strategy in order to produce meaningful road patterns. The controlled generation of procedural content was also studied by Lipp et al [LSWW11]. Their method allows an efficient and controlled change of the appearance of an existing layout by the definition of layered constraints. These constraints include relative positioning of parcels, city blocks, or streets to user-defined anchor points. The layout is updated by merging the content defined on different layers into the existing city layout, while locally preserving defined constraints.

## 2.2 Techniques for City Blocks Layouts Synthesis

City blocks, i.e., the regions within a street network completely enclosed by roads, represent an essential part of every city layout. They can be depicted as atomic building entities that are surrounded by streets and offer the space for different types of entities such as buildings, recreational areas, or office zones. For the planning process of new urban districts or the re-development of existing ones, they are indispensable. For virtual worlds, they are as important as for real cities as they offer space for virtual assets and thus make the virtual space lively. In general, the computation of a city-block layout can be conducted in two different ways, when the current state of the art is reviewed. The first possibility is subdividing the empty space inside the block into smaller regions called parcels. Individual buildings might be later placed into these regions. The second approach abstracts real-world objects by polygons with optional attributes. These polygons serve as input to a custom-tailored layout algorithm that either uses statistics or practical guidelines as rules for the composition of a new city block layout. In many cases, the layout is computed using a stochastic optimization technique such as a Markov Chain Monte Carlo (MCMC) sampler that discovers layouts having a high probability according to the evaluation of a problem tailored objective function.
In the following, we first review techniques that focus on the subdivision of city

blocks into parcels and then advance to studying more general layout algorithms in the second part of this section.

### 2.2.1 City Block Layouts using Subdivision Schemes

Parish and Müller [PM01] already discussed an early approach for the subdivision of the space encompassed by a city block. Their algorithm generates parcels by recursively splitting the city block along longest and near parallel edges into polygonal regions that are called parcels. During the recursive splitting, they restrict the algorithm to only produce convex parcels. The subdivision into smaller areas and finally parcels is stopped when a certain minimum area falls below a certain user-defined threshold.
Aliaga et al. [AVB08] follow a different strategy to subdivide the city block into parcels. They compute the medial axis of the city blocks, approximated by the major axis of the object-oriented bounding box (OBB). Points along this major axis are sampled according to learned statistics. These points are then duplicated and randomly perturbed to form sites that are used to derive a Voronoi diagram. The intersection of the Voronoi edges with the outline of the city block produces an interior skeleton that splits the space into plausible parcel regions. This strategy even leads to a likely parcel layout, when the shape of the city block outline is non-convex.
Vanegas et al. [VABW09a] focus on the procedural extension of existing urban spaces and demonstrate their method by simulating the future development of urban areas using socio-economic data. In order to produce meaningful parcel layouts within newly generated city blocks, they extend the simple recursive subdivision scheme of Parish and Müller. More precisely, they start by computing the (OBB) of the city block polygon. In a subsequent step, a simulation step predicts the number of parcels to be generated within the current city block polygon. This information is utilized to decide whether to introduce a new street segment or a parcel boundary, by the split along the minor axis of the OBB. The splitting algorithm repeats the prediction and the subdivision step until a desired number of parcels is reached.
Vanegas *et al.* [VKW$^+$12] generalizes parcel generation within city blocks, by introducing two different subdivision schemes. The first one is based on the recursive splitting scheme introduced by Parish and Müller [PM01]. It uses an OBB based splitting mechanism as already proposed in Venegas et al. [VABW09a]. The major difference between Vanegas et al. [VABW09a] and Parish and Müller [PM01] is that the OBB is computed to be aligned with an existing parcel edge leading to a more robust subdivision. In addition, their approach can control the street access of parcels, by splitting the current parcel either along the major or minor axis according to a user defined probability. Their second approach computes the me-

dial axis of the city block polygon, which is approximated by the straight skeleton [AAAG96]. Such a skeleton divides the city block into several faces where each face is adjacent to an outline edge of the city block polygon. However, these faces contain undesirable diagonal edges near street intersections resulting from angular bi-sectors. A re-assignment of such faces to adjacent regions removes these artifacts depending on street direction and street-length. The resulting 'cleaned' regions are subdivided into parcels, by casting rays from sample points drawn along the street segments towards the adapted city block skeleton.
Yang *et al.* [YWVW13] use templates with predefined parcel layouts. These templates are similar to their street pattern templates that were already discussed in section 2.1.2. Parcel templates in their approach contain different patterns such as strips, i.e., multiple parcels placed next to each other, and spikes, i.e., parcel layouts that are modeled around a dead-end road segment. These templates are assigned to regions and might be procedurally expanded in order to fill up empty space. This is especially used to achieve a tight parcel packing within the city block. The candidate parcel layout is then non-linearly deformed to match the shape of the region where it was assigned to.

**Commercial Software for City Block Layouts** Apart from approaches proposed by the scientific computer graphics community, different commercial software packages provide tools that are suited for the layout design of city blocks. Civil 3D from Autodesk [Sof17] and SiteOps [Ops17] provide interactive and semiautomatic tools for parcel design. Both software packages provide algorithms for computing parcel layouts along street segments by splitting the segment by point sampling. From these points, rays orthogonal to the street segments are traced up to a certain distance to produce the shapes of the parcels. The user is then able to manually adapt the layout until his demands are satisfied.
CityEngine from Esri [Esr17] provides a set of parcel layout algorithms inspired by recent techniques from the computer graphics research community. Early versions of the software use an algorithm adapted from Parish and Müller [PM01]. More recent releases use adoptions of the algorithms proposed by Vanegas et al. [VKW+12]. They even provide a combination of the strip-based layout and the adaptive subdivision algorithm mentioned in the paper of Vanegas et al..

### 2.2.2 Generation of General Discrete Element Layouts

Orthogonal to the previous discussed methods are discrete element layout algorithms that use stochastic optimization techniques, *e.g.* Markov Chain Monte Carlo (MCMC). These methods learn object relations from a set of examples [YYT+11], are feed with well-known design guidelines [MSL+11], or use object

relations encoded in factor graphs [YYW+12, YBY+13] in order to synthesize novel element layouts.

## 2.3 Techniques for Sequence Generation and Building Synthesis

In here, we discuss techniques for the synthesis of buildings and architectural structures using procedural as well as example-based approaches. Before we discuss techniques that can be used for the generation of buildings we discuss graph-based optimization techniques. These techniques arrange exemplars into sequences by minimizing problem specific objective functions. Such optimization techniques are important because a large variety of content within the computer graphics community is arranged as one-dimensional sequences. Concrete examples are audio data, captured motion data, or video sequences. Even images can be transformed into 1D sequences by slicing it into patches along one image dimension. Similar to images buildings and architectural elements such as ornaments can be interpreted as 1D sequences of smaller entities that are arranged along a primary dimension. A discussion of related graph-based optimization techniques is mandatory as the work at hand proposes a novel optimization technique in section 3 that is applied to the application of building synthesis in chapter 6. In addition, one algorithm for the synthesis of footprint layouts within city blocks in chapter 5 utilizes such an optimization technique.
We start with a discussion of these types of algorithms and then move on to more specific techniques for the generation of architectural structures and buildings.

### 2.3.1 Fixed-Sized Sequences

Graph-based optimization techniques can be split into two different categories. The first category produces sequences that are composed of a fixed number of elements, while the number of elements present in the result is known in advance. In addition, all exemplars or elements have the same elongation in e.g. seconds, video frames, pixels, or meters. For the example-based synthesis of audio tracks, Wenner et al. [WBSH+13], decompose audio data according to beat analysis into smaller chunks of fixed size, ranging from 250 ms to 1500 ms depending on the detected beat of the music genre. A novel song is synthesized by defining a target length and the subsequent computation of a sequence that consists of a fixed number of elements that is computed from the target length and the fixed size audio chunks extracted from an exemplar song. The audio chunks that compose the novel song are determined using a dynamic programming approach similar to the

one presented by Viterbi [Vit67]. Zhou et al. [ZLL13] synthesize texture patterns along curves. The input to their algorithm is a parametric curve and a texture patch containing the pattern that is used during the synthesis step. In a first step a ribbon around this curve is constructed by computing offset curves using positive and negative normal direction. The domain enclosed by the ribbon is split into a fixed number segments, when is curve is split into a fixed number of equally size arc length regions. A dynamic programming algorithm is used to determine a sequence of texture regions that are optimally aligned with the ribbon segments and additionally minimizes the error in color transitions between neighboring ribbon segments.

### 2.3.2 Variable-Length Sequences

Lefebvre et al. [LHL10] propose an algorithm to compose new images utilizing image strips of varying size. Their algorithm is able to compose a new image of a fixed size, without having to specify the number of image strips used in the final result in advance. Recently, Zhou et al. [ZJL14] proposed an algorithm for the synthesis of structured vector patterns that follow a predefined curve. The input to the algorithm is a vector pattern composed of polygons with properly defined interior and exterior areas, which is split into smaller parts using parallel slices. For each slice, a topology descriptor is computed. The method uses a constrained dynamic programming approach in order to recombine the slices by using topology descriptors to compute a topology-aware layout. A continuous optimization scheme refines the layout be ensuring continuous connections between individual parts and produces a vector pattern ready for fabrication.
In computer animation, various shortest-path algorithms have been proposed to synthesize novel motions from motion capture databases. Kovar et al. [KGP02] presented a greedy branch-and-bound approach to generate walks through a motion graph following a predefined path. Their ideas were extended by Safonova and Hodgins [SH07], who introduced interpolated motion graphs and an efficient $A^*$ search to generate smoother and more complex animation sequences along a path that may include environmental constraints similar to our key points described in Section 3.3.1. Lo and Zwicker [LZ10] reduced the exponential search space of this kind of problem, by employing a bidirectional $A^*$ algorithm and merging the two search trees by interpolation, they were able to synthesize human motions at interactive rates.
In order to compute these sequences along a path, the surrounding space is discretized. In the case of Safonova and Hodgins [SH07] motions segments are placed at the start point of the path. The space around the path is discretized into grid cells. From one or multiple segments, a graph is grown and dynamically expanded using variants of Dijkstra or A* shortest path computation. The space

around the path is discretized into $d$ cells and a database of $n$ motion segments is used. In order to compute path that consists of $d$ motion segments out of $n$ motion segments the search space is already $O(n^d)$. This search space was reduced by Lo and Zwicker [LZ10] using a bi-directional algorithm that expands one search tree from the start point towards the endpoint and a second search tree from the endpoint towards the start point. This drastically reduces the search space from $O(n^d)$ to $O(n^{\frac{d}{2}})$.

The algorithms of Lefebvre et al. [LHL10] and Zhou et al. [ZJL14] synthesize one-dimensional structures by discretizing the space along an image dimension or along the parameter domain of a curve. They further require that the generated object satisfies a certain image size (Lefebvre) or length and topology (Zhou). Both algorithms construct a graph from a defined start point towards a defined endpoint by iteratively adding nodes. Thus, there search space is in case of $d$ discretization steps and $n$ elements in $O(n^d)$. After a new node is added to the graph, the paths are re-ordered by the optimization algorithm using the transition cost. The online graph construction might be inefficient in cases when a large number of paths is generated that will not lead to a solution that satisfies the requirements. Thus in the worst case, the runtime of such types of algorithms might be in $O(n^d)$. The algorithms of Zhou et al. [ZLL13] and Wenner et al.[WBSH+13] reduce the problem complexity by specifying the number of elements within the resulting sequence in advance.

### 2.3.3 Resource Constrained Shortest Path

Although this NP-hard [GJ79] problem has been studied for more than four decades and several pseudo-polynomial algorithms have been proposed, we are not advocating for any of the existing RCSP algorithms for several reasons. Most of them set only an upper bound on the constraints [Zie04, ZW12]. In our applications, however, also a lower bound is necessary and needs to be fulfilled. It makes the problem much harder, even for the single resource case, as it includes the well-known knapsack problem with an equality constraint as a special case ([MR84, GJ79]). The algorithms of Ribeiro and Minoux [RM85] and Turner [Tur12] are designed to deal with both upper and lower boundaries. However, the method of Ribeiro handles only one constraint and the work of Turner only discusses equality constraints. We aim to provide multiple solutions, strictly speaking, the $k$-best solutions, that was not taken into account and discussed by previous approaches.

### 2.3.4 Synthesis of Buildings

Next, we discuss synthesis schemes for individual buildings. The seminal city generation system of Parish and Müller [PM01] proposed the generation of buildings using stochastic L-systems. Subsequent manipulation of the ground plan generates concrete building instances. Different L-system modules encapsulate operations such as extrusion, transformation, and branching, while others encapsulate geometric templates of building parts. String outputs produced by the L-system are transformed into concrete building geometry, that is further enhanced with procedural textures that are organized as layered grids. The parametric generation of buildings using grammar rules was extended by Wonka et al. [WWSR03] proposing a parametric set shape grammar to describe the construction of buildings. This overcomes the difficulty with L-systems that are typically used to grow open spaces such as road networks. The split grammar starts with an initial shape and splits the different facades into floors down to individual windows and ornaments. The limitation to simple splits was lifted by Müller et al. [MWH+06] who contribute the 'repeat split' rule and the adaptive scaling of rules. The proposed CGA shape grammar was extended by Schwarz and Müller [SM15] to prioritize the shapes and enable direct access to shapes and the derivation tree. Both systems are integrated into the CityEngine [Esr17], the commercial city planning toolchain by Esri.

In order to overcome the manual definition of rules, Bekins and Aliaga. [BA05] propose Build-by-Numbers, a technique for the time-efficient design of buildings. From a set of images, an initial textured 3D model is reconstructed. The user is asked to subdivide the reconstructed mass model into different features, i.e., floors, facade, windows, doors, and decorations. The system analyses the various labels and automatically derives rules for the repetition and reorganization of existing architectural elements. Their work is a very early approach to inverse procedural modeling, i.e., the process of automatically determining the production rules. Müller et al. [MZWVG07] focus on the derivation of a meaningful facade hierarchy in order to overcome the manual definition and grouping of semantically similar architectural elements. Automatically deriving construction rules for 3D geometry was addressed by Bokeloh et al. [BWS10]. They derive construction rules for building geometries by exploiting self-similarities and symmetries within a given 3D model. For procedural editing of point clouds similar to the early work of Bekins and Aliaga [BA05] was studied by Demir et al. [DAB15]. For additional details about inverse procedural modeling techniques apart from buildings we refer the interested reader to the recent tutorial of Aliaga et al. [ADBW16] that provides an in-depth discussion of the state of the art.

In contrast to procedural algorithms, example-based techniques have also been proposed for generating man-made objects and architectural structures. One of

the first approaches was the model synthesis technique introduced by Merrell [Mer07]. This technique draws ideas from texture synthesis literature (see Wei et al. [WLKT09] for a detailed overview). The algorithm expects as input a 3D model decomposed into smaller pieces, that serve as building blocks. These can be recombined or placed next to each other, guided by a set of Boolean functions that encode adjacency rules. The model synthesis is realized by a labeling algorithm that operates on a 3D lattice and assigns building block labels to each lattice cell without violating the adjacency constraints. Merrell and Manocha [MM08] extended the model synthesis algorithm to work on a continuum instead of a grid. The adjacency constraints are reformulated to operate on local neighborhoods. Thus, a synthesized model, when viewed at a microscopic level, resembles the style of the exemplar. Strictly speaking every local neighborhood within a synthesized model needs to be present somewhere within the exemplar. In order to improve the flexibility of the model synthesis technique, additional constraints were added in Merrell and Manocha [MM09] to capture the intent of the user. In addition to the simple adjacency constraint [Mer07, MM08], algebraic constraints and connectivity constraints can be specified to control the output of the algorithm and to produce a large variety of output models from one or multiple input exemplars. A generalization of the model synthesis algorithm [Mer07] towards tile patterns was proposed by Yeh et al. [YBY$^{+}$13]. Their approach is inspired by MC-SAT, a technique that mixes Markov Chain Monte Carlo (MCMC) techniques with satisfiability problems and models a pattern as a probabilistic graphical model called factor graph. Using such a model enables to integrate adjacency constraints with more than two model pieces into the synthesis step.
The more complex the examples that should be synthesized or re-targeted, the more complex is the challenge of capturing the intent of the user. The task gets even more complicated when irregularities are present that need to be handled by the synthesis algorithm. Re-synthesis of irregular structures on the special case of irregular buildings is proposed by Lin et al. [LCOZ$^{+}$11]. The model is transferred to a more abstract representation of axis-aligned bounding boxes that envelope geometric details at multiple levels. These boxes are used to construct a hierarchy to capture large-scale structures. Within these bounding boxes, dominant structures are identified by the computation of longest sequences sharing similar attributes. Resizing a model is then iteratively achieved by repeating or scaling bounding boxes that are captured by dominant sequences. A similar approach was presented by Wu *et al.* [WLW$^{+}$14]. Instead of a manual decomposition of the input model, a set of principal re-targeting directions is derived. In addition, descriptors to measure self-similarities are computed. Resizing a model is achieved by replication of the original model inside a 3D grid. Thus multiple model parts are assigned to each cell that are differentiated by a unique label. Similar to Merrell [Mer07] a labeling problem is solved. Instead of the model synthesis algorithm, a

multi-label graph cut optimization technique is used to compute a smooth labeling within a 3D grid. The final labels assigned to the individual grid cells are used to reconstruct the resized 3D model from different parts of the replicated model instances.

# CHAPTER 3

# Efficient Optimization with Multiple Resource Constraints

In this chapter of the work at hand, we describe an efficient algorithm for example-based content generation. Its heart is a novel multi-constrained optimization technique, which generalizes the resource constraint shortest path (RCSP) problem in order to produce a portfolio of multiple optimal solutions. The content of this chapter is based on the peer-reviewed publication

> Stefan Hartmann, Elena Trunz, Björn Krüger, Reinhard Klein, and Matthias B. Hullin. Efficient multi-constrained optimization for example-based synthesis. *The Visual Computer / Proc. Computer Graphics International (CGI 2015)*, 31(6-8):893–904, June 2015.

## 3.1 Motivation

An important sub-class of problems that occurs in virtually all fields of multi-media computing: the arrangement of atomic elements from a database into a sequence that is optimal under some problem-specific objectives and constraints. Whether it be animation sequences, architectural models, audio chunks, video snippets, many types of media content are composed of atomic elements along a one-dimensional parameter domain. Concrete examples include duration or path length of human motions, duration of audio and video sequences, building length, or building height of individual architectural models. Examples for atomic building blocks used to compose different media types are single images, audio snippets, collections of human motion snippets, image slices, or even building parts. Typically, the various atomic elements carry multiple attributes and relations (transitions) to other elements that are encoded and stored inside a concrete element

database. Possible transitions between the different elements form a graph that can be traversed in order to produce novel content that is composed of multiple atomic elements. However, a typical goal is to compute a sequence of elements that is optimal according to an application specific objective function and in addition satisfies specific constraints such as length, element occurrence etc. Thus, it is not enough to compute arbitrary paths through the mentioned graph, but we are strongly interested in computing one or multiple paths that minimize the objective function, while none of the requested constraints is violated. In this chapter, we propose to formulate the resource-constrained synthesis of new sequences from atomic elements as a general resource-constrained $k$-shortest path (RCKSP) problem. For the $k = 1$ case (RCSP), it has been shown that it abstracts some problems in graphics well. The more general task of finding multiple optimal solutions, on the other hand, has received much less attention although it corresponds to problems where a portfolio of candidate solutions is desired. Both features are of great relevance to content generation problems in computer graphics. A unique property of our approach and the key to its superior performance is that the graph traversal is guided by resource use, rather than the cost function as is usually the case. This turns out to be a significant advantage if the number of elements in the database is large. In addition, our technique is capable of meeting an extended set of goals: (a) first, to satisfy multiple constraints which might be intervals; (b) second, to find $k$ guaranteed optimal solutions; (c) third, generalized structures where elements can have more than two neighbors such as t-junctions.

## 3.2 Resource-Constrained Sequences

In the following, we formulate the problem of computing optimal sequences subject to multiple constraints (equality and interval) with the focus on computing a set of $k$ optimal solutions. In addition, we will compare the different features of other state of the art algorithms (see section 2.3) in solving the special case of $k=1$ (see Table 3.1).

**Optimal Sequences as Resource-Constrained Shortest Paths.** Assume a database $D$ containing $n$ elements. Allowing multiple occurrences of the same element, we compose a sequence $X$ of elements from $D$. Every time element $e$ is used, it consumes a certain amount of resources as given by its non-negative *resource usage vector* $\mathbf{r}(e) = (r_0, ..., r_m)(e)$, $m$ being the number of different resources in our budget. The cost of placing $e_i$ immediately before $e_j$ in a sequence is given by the *transition cost function* $\delta(e_i, e_j) \geq 0$. The total resource usage, $\mathbf{R}$, and the total transition cost, $\Delta$, for a sequence $X = (e_1, ..., e_p)$ of length $p$ add up as

$$\mathbf{R}(X) = \sum_{i=1}^{p} \mathbf{r}(e_i) \quad \text{and} \quad \Delta(X) = \sum_{i=1}^{p-1} \delta(e_i, e_{i+1}). \tag{3.1}$$

Let $\chi_c$ be the set of all *feasible* sequences that consume resources within certain given bounds,

$$\chi_c = \{X \in \chi \mid R_i^{\min} \leq R_i(X) \leq R_i^{\max} \; \forall \; i = 1, ..., m\},$$

where $\chi$ is the set of all sequences of elements of $D$. Our objective now is to find $\chi_{\text{opt}}^k = \{X_{opt}^1, \ldots, X_{opt}^k\}$, the set of $k$ solutions, each of which is both feasible, i.e., it satisfies the user defined resource constraints, *and* optimal with respect to the total transition cost after exclusion of known better solutions. In other words, the $j^{\text{th}}$-best solution $X_{opt}^j$ can be defined recursively as follows:

$$X_{opt}^j = \underset{X \in \chi_c \setminus \{X_{opt}^1, ..., X_{opt}^{j-1}\}}{\operatorname{argmin}} \Delta(X) \tag{3.2}$$

The constraint vectors $\mathbf{R}^{\min} = (R_1^{\min}, ..., R_m^{\min})$ and $\mathbf{R}^{\max}$ may be a mix of integers and real values, all non-negative. Their meaning depends on the application and can include the length of a generated sequence as well as other parameters (*"minimum number of balconies on city block"* or *"number of jumps in motion sequence"*). For now, we note the presence of both lower and upper bounds, which define an interval or an equality constraint if set to the same value (e.g., *"total duration of media playlist in seconds"*). Following the example of Lefebvre et al. [LHL10], we interpret the above configuration as a graph. The nodes of the graph are elements $e$ from database $D$ consuming a fixed resource usage vector $r(e)$. The transition between subsequent elements is described by the cost $\delta(e_i, e_j)$ that represents the weight of the directed edge from state $e_i$ to state $e_j$. The problem of minimizing the cost of a sequence as per Eq. 3.2 then immediately translates to computing the shortest path through the graph under the given resource constraints, a class of problem that has been investigated in operations research [RM85, Gar09, Tur12].

| Method | Resource constraints / type | Key points / type | 1.5D |
|---|---|---|---|
| Safonova[SH07] | None | **Yes**/Class | No |
| Lo[LZ10] | None | **Yes/Class** | No |
| Lefebvre[LHL10] | One/Equality | Yes/Element | No |
| Wenner[WBSH$^+$13] | One/Equality | **Yes/Class** | No |
| Zhou[ZLL13] | One/Equality | No | No |
| Ribeiro[RM85] | One/Interval | No | No |
| Ziegelmann[Zie04] | Multiple/Upper | No | No |
| Zhu[ZW12] | Multiple/Upper | No | No |
| Turner[Tur12] | **Multiple/Equality** | No | No |
| Zhou [ZJL14] | **Two/Equality** | No | No |
| **This work** | **Multiple/Interval** | **Yes/Class** | **Yes** |

Table 3.1: A detailed comparison of the algorithmic features that are supported by algorithms that were discussed in detail in section 2.3. The feature comparison includes the type of constraints the different algorithms support, the handling of keypoint elements, i.e., the enforced placement of a specific element at a certain position and the support of 1.5D structures. These type of structures allows using elements that can be connected with more than two other elements. This is especially necessary when the provided input that should be serialized is similar to a tree (see Figure 3.3).

## 3.3 Efficient Resource-Constrained Sequence Optimization

### 3.3.1 Single Resource-Constrained Optimization

In this section, we describe and formulate a multi-constrained optimization approach for example-based sequence generation. The heart of our approach is an *intermediate graph* representation, which serves two major purposes. First, it transforms the original constrained problem into an unconstrained one. Second, it spans the search space, containing only feasible solutions, i.e., it only contains solutions that satisfy the constraints independent from the transition cost. To this end, we introduce discrete *resource consumption levels* (henceforth simply referred to as *level $l$*) that group subsequences of elements by their resource usage and impose a total order on the intermediate graph. After the intermediate graph has been built, the transitions costs are computed and assigned to the edges connecting consecutive elements. Then we have several options. First, we can com-

pute the optimal solution according to the transition cost, using the well-known shortest path algorithm, e.g., Dijkstra's algorithm. Second, we can also compute a larger class of $k$ solutions including the optimal one and $k-1$ other near-optimal solutions, sorted by increasing total transition cost. Such a set of solutions can be computed using algorithms for computing $k$-shortest paths, for example, the one proposed by Eppstein [Epp94] .

**Single-Resource Case.** Let us first discuss the construction of the intermediate graph for problems that underlie a single constraint, leaving us with a scalar resource budget. The satisfaction of equality constraints is a major strength of our algorithm, so we will for now focus on this type of constraint. The intermediate graph is *directed* and *acyclic*, properties achieved by unrolling re-occurrences of elements in a sequence. Its nodes are represented as $N(e,l,r^c)$, where $e$ denotes an element from the database, $l$ represents the level on which the node will be placed, and $r^c$ tracks the already allocated resource so far, including the element's own $r(e)$. Construction of the intermediate graph happens in three stages: *initialization*, *growing*, and *pruning*.

*Initialization:* Before the actual intermediate graph is built, first the number of levels $L$ is determined from the single constraint $R$. In our setting, a constraint represents an application specific attribute $A$ that needs to be satisfied by all feasible solution sequences in the set $\chi_c$. The number of levels $L = R/GCD$ is determined by dividing $R$ by the greatest common divisor (GCD) that the exemplars of $D$ share for that specific attribute. The value of $R$ is here derived from the specified maximum allowed resource consumption $R^{max}$. For real-valued constraints, if no GCD exists, we rescale/round the elements' resource usage values so that a GCD can be computed. In this way, the real-valued resource is transformed into one that consumes integer values and can be used in the synthesis setting. We perform this as a preprocessing step and store the result for each element $e \in D$. For the sake of simplicity, let us assume $GCD = 1$ for the time being. Now, elements $e_i \in D$ that are allowed to be placed at the front and at the back of the solution sequences $\chi$ are inserted into the yet empty graph at levels $l_f = 0$ and $l_b = L - r^c(e_i)$. Their nodes are set to carry the information $N(e_i, l_f, r^c(e_i))$ for front elements and $N(e_i, l_b, L)$ for elements at the back, and they are connected by zero-cost edges from a source node $s$ (front elements) or to a sink node $t$ (back elements). The level can now be interpreted as a position between source and sink.

*Graph Growing:* We grow the graph in a bi-directional fashion by alternating between *forward* expansion from the source $s$ (listed in Function 1) and *backward* expansion from the sink node $t$, until a stopping criterion is met. In order to make the growing process easy to understand, the pseudo code presents the procedure in its simplest form. In the following, we explain its efficient implementation using

```
for all nodes v1 = (e, l_st, r^c(e)) do
   for all possible successors e_succ of e do
      l_succ ← l_st + r(e);
      r^c(e_succ) ← r^c(e) + r(e_succ);
      if l_succ ≤ l_ts and r^c(e_succ) ≤ L then
         v2 ← (e_succ, l_succ, r^c(e_succ));
         if node v2 not exists then
            create node v2;
         end if
         create edge (v1, v2, δ(e, e_succ));
      end if
   end for
end for
```

Function 1: Construction of the intermediate graph (forward step)

two separate priority queues $Q_f$ and $Q_b$, one for each expansion direction.
The fundamental difference of our algorithm when compared to other graph search algorithms is that the nodes in $Q_f$ and in $Q_b$ are sorted by the current resource consumption instead of the path cost. More specifically, $Q_f$ will prioritize nodes with the *least* resource consumption $r^c$, while $Q_b$ processes with the *highest* consumption $r^c$ first.
Without loss of generality, we start explaining the forward step. In order to expand the graph towards $t$, we extract the current top node $N_f(e_f, l_f, r^c(e_f))$ from $Q_f$, generate all possible successor nodes $N'(e_{succ}, l_{succ}, r^c(e_{succ}))$, and connect them with an edge of cost $\delta(e_f, e_{succ})$. The level $l_{succ}$ of node $N'$ is computed from the current top element of $Q_f$, namely $l_{succ} = l_f + r(e_f)$, and the overall resource consumption is increased by the successor's usage, $r^c(e_{succ}) = r^c(e_f) + r(e_{succ})$. The backward expansion works along the same line, producing nodes that precede those currently stored in $Q_b$. Each time this step is executed, the current top node $N_b(e_b, l_b, r^c(e_b)$ of $Q_b$ is extracted and predecessor nodes $N'(e_{pred}, l_{pred}, r^c(e_{pred}))$ are generated, adding edges with cost $\delta(e_{pred}, e_b)$. Here, $e_{pred}$ is an element from possible predecessors of $e_b$ and its node level $l_{pred} = l_b - r(e_{pred})$ is computed by simply subtracting the resource usage $r(e_{pred})$ of the predecessor element $e_{pred}$ from the current node level $l_b$. The resource consumption so far is updated accordingly, $r^c(e_{pred}) = r^c(e_b) - r(e_b)$. In case if a node $N'(e, l, r^c(e))$ already exists only an edge with assigned cost $\delta$ is inserted into the graph (see Function 1). We label nodes by the direction in which they were created (forward, backward) and restrict both steps to only expand nodes that were created from the particular expansion step. The algorithm terminates when the level of the current top node $l_f$ overlaps with the level $l_b$ of the current top element in $Q_b$, or if both queues are empty. We

give a detailed complexity analysis in section 3.4. There, we prove that the worst-case complexity of the proposed algorithm for a single constraint is in $\mathcal{O}(Ldn)$, where $n$ denotes the number of elements in the database and $d$ is the maximum count of concatenation neighbors determined across all elements.

**Pruning.** In order to reduce the nodes that might be traversed during the optimization, we apply two different pruning strategies: *overflow/underflow* pruning and *dead-end* pruning. The first strategy focuses on detecting resource overflows/underflows. Thus nodes that would produce a resource overflow/underflow will not be created during the expansion step, as they would not be part of any feasible solution at all. The second strategy scans the graph for existing nodes that do not have any successor or predecessor nodes, i.e., dead-end nodes. These nodes would also not become part of any solution. Thus, we remove them from the intermediate graph too.

### 3.3.2 Extension to Multiple Resource Constraints

We now extend the single-resource solution to satisfy multiple equality constraints simultaneously. First, we revert from using the scalar resource usage $r$ to the vector $\mathbf{r}$, ending up with a node definition of $N(e,l,\mathbf{r}_1^c(e))$. When incorporating multiple constraints, the total order implicitly given by the single scalar resource consumption is lost. In order to re-establish a total order, the generated nodes are inserted lexicographically into the queues using their attached vectors tracking the currently consumed resources $\mathbf{r}^c(e)$). During the initialization, the number of levels is determined over the sum of all constraints ($L = \sum_{j=1}^{m} \mathbf{R}_j/GCD_j$), where $GCD_j$ is the greatest common divisor that the exemplars of $D$ share for the attribute $j$. The levels of the nodes are now computed by adding up the resources, arriving at $l_{succ} = l_f + \sum_{i=0}^{m} r_i(e_f)$ for the forward step. To keep track of consumption of each individual resource, the currently consumed resource vector $\mathbf{r}^c$ is updated as $\mathbf{r}^c(e_{succ}) = \mathbf{r}^c(e_f) + \mathbf{r}(e_{succ})$. Just as before, for the backward expansion step, we subtract the resources accordingly. Unlike in the single-resource case, multiple instances of an element may occur on a level, since different combinations of resources may result in the same sum. However, although multiple instances of a specific element might be placed on the same level, they can still be distinguished from the other instances, because of their unique vector $\mathbf{r}^c(e)$ tracking the resources consumed until that element instance so far. Thus a new node is only generated if no node having the unique node signature $N(e,l,\mathbf{r}^c(e))$ already exists. If such a node $N(e,l,\mathbf{r}^c(e))$ already exists, only an edge is inserted. Further, we note that the relative scales of the resource components, and their order in the vector, have no influence on the final outcome of the algorithm. They do,

however, affect the intermediate graph, the order in which the solutions are generated, and possibly the memory or runtime requirement of our algorithm. The construction procedure for a small example that only contains three elements is illustrated in Figure 3.1.

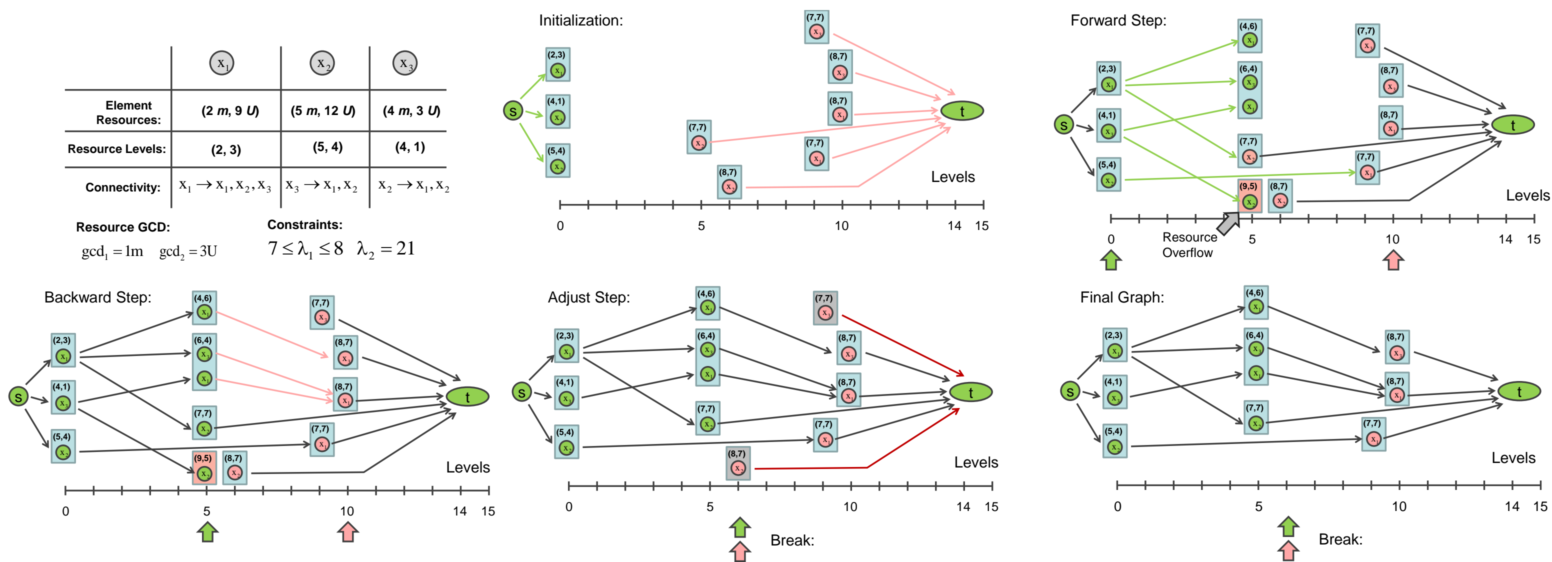



Figure 3.1: Illustration of the graph setup procedure for a small example consisting of three elements. The constraints that need to be satisfied are one equality constraint $\lambda_2$ and an interval constraint $\lambda_1$.

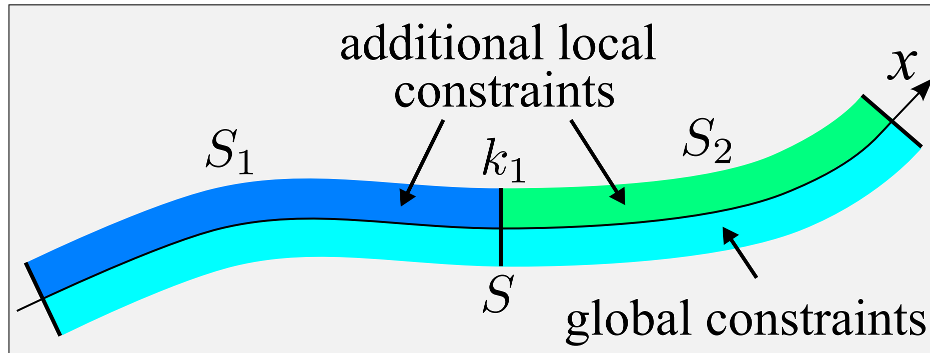


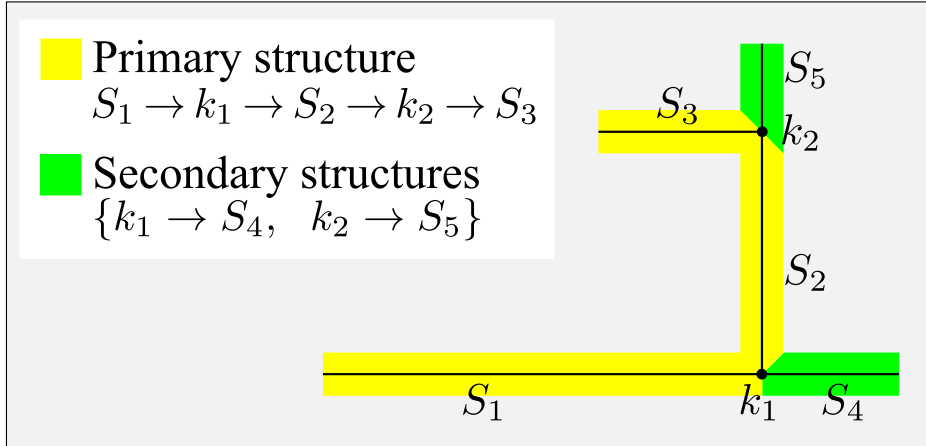

Figure 3.2: Structure is a set of constraints defined on the primary parameter domain, $x$. Here, the key point $k_1$ divides the structure $S$ into substructures $S_{1,2}$ that may each be constrained differently.

### 3.3.3 Interval Constraints

Our method is also able to handle the request for interval constraints. In order to support such constraints, the graph construction procedure does not need to be changed. The only step that needs to be adapted is the second part of the initialization, which was described in Section 3.3.1. Instead of a single end level $L$, we now have an interval between $L_{min}$ and $L_{max}$. Nodes for the elements allowed to be placed at the back of the solution sequences are then created for *all* levels from $L_{min}$ to $L_{max}$. We observe that, despite the backward graph being expanded from a multitude of root nodes, in practice, this bloats the graph less severely than expected. After all, during the expansion step, elements already present on a level can be re-used and will not need to be duplicated.

### 3.3.4 Extension to Higher Order Sequences

A *sequence* in the sense of this chapter is a succession of elements, which we associate with an arrangement along a primary parameter $x$, such as length of an object or time to move from a start point of a curve to the endpoint of that curve. Our algorithm treats this primary parameter as one out of several resources of which each element uses different amounts. Hence, if a certain time or length needs to be filled, the resulting sequence could consist of many short elements or just a few long elements. In order to give the user precise control over the synthesis, we use the concept of *structure*, which we define in a narrower sense than usual. The structure is the entirety of constraints on the synthesis.
It is specified by the user at and in between *key points* that live on the primary parameter domain of the content modality. For instance, the structure of an ani-




Merged structure:

$S_1 \rightarrow k_1 \rightarrow S_4 \rightarrow k_1 \rightarrow S_2 \rightarrow k_2 \rightarrow S_5 \rightarrow k_2 \rightarrow S_3$

Figure 3.3: We decompose binary tree structures by recursively extracting the longest paths from the structure graph.

mation might contain the specification, “the actor should jump at time 5 seconds”. It is not until the final sequence has been assembled that this key point will correspond to some $i^{\text{th}}$ element that happens to end up at that particular time. The connection between structure and the sequence is, therefore, an implicit one. In this section, we will describe how our algorithm handles structural constraints in terms of key points, and how they can be used to assemble generalized (“1.5D”) structures.

**Key points** divide an 1D structure into a set of *substructures* $S = \{S_1, \ldots, S_{m+1}\}$, where $m$ is the number of key points (see Fig. 3.2). They may further enforce the localized occurrence of a specific element *class*, which is a harder task than just enforcing a particular element because it means that adjacent substructures cannot be treated independently of each other. Additional local constraints may be specified for the domain covered by each of the substructures $S_i \in S$. For each $S_i$, the algorithm constructs a *subgraph* $G_i$, where elements placed at key points serving as front or back elements. All subgraphs $G_i$ are now merged into the ’global’ intermediate graph $G$, while the front elements of $S_0$ are connected by zero-cost edges to the source node $s$ and the back elements of $S_m$ connected by zero-cost edges to the sink node $t$.

**1.5D Synthesis.** Our framework described so far can also be used to synthesize structures in the shape of a binary tree or closed curves, while still guaranteeing

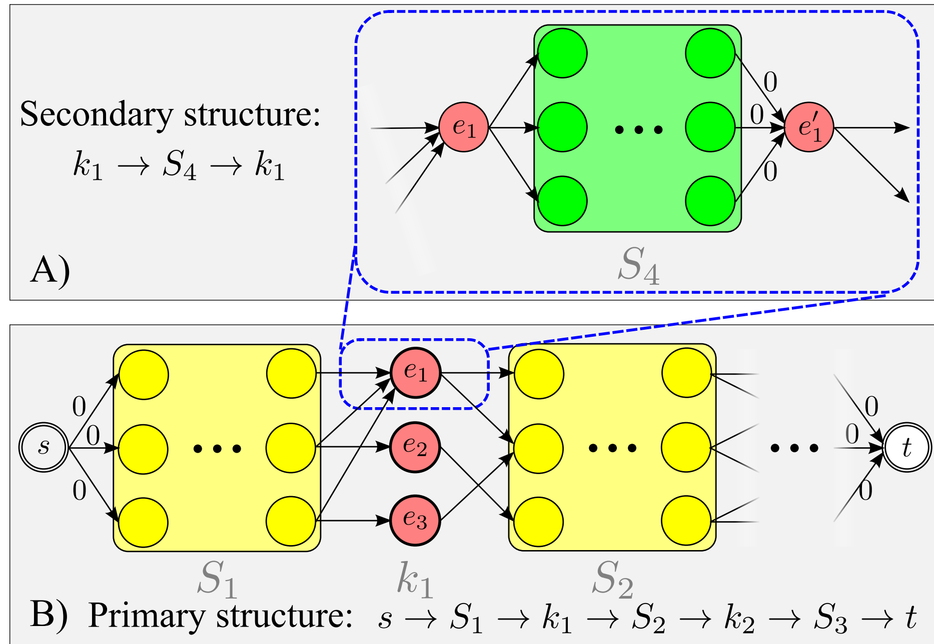



Figure 3.4: Merging the subgraph of a child structure into its parent.

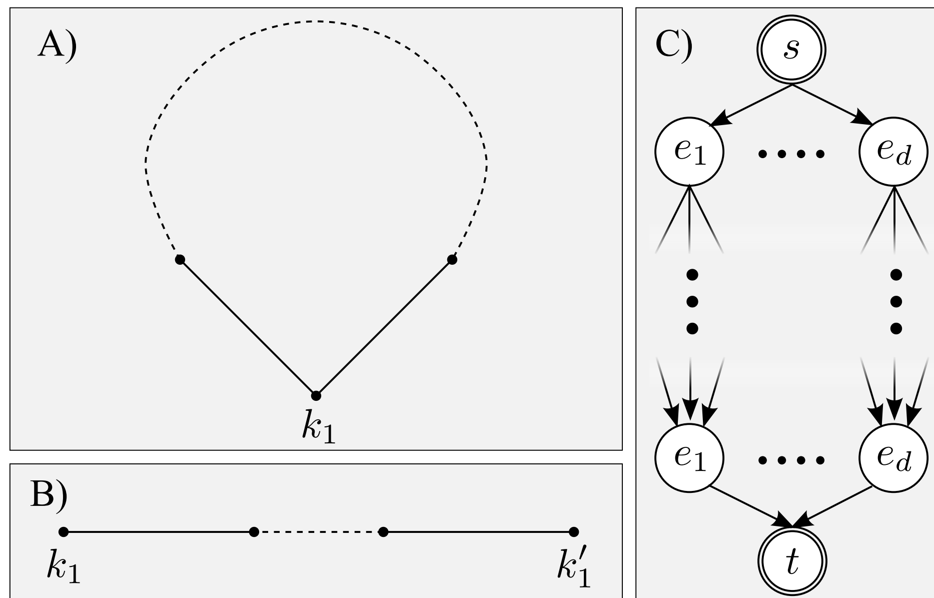



Figure 3.5: Closed curves are split at a key point that are cloned, thus, serving as start and end element of a sequence.

a globally optimal result. To allow branching, we introduce a special type of key point with three neighboring elements that represent the branching elements inside a binary tree. The major challenge here is to transform the tree into a 1D sequence and ensure the uniqueness of the elements that will end up being placed at key points. We solve this by performing an iterative longest path search starting from the root node of the initial tree (Figure 3.3). We remove the path edges from the tree and store them as the *primary* structure. From the remaining connected components, we extract a set of *secondary* structures, and so on, until no more connected components are found. We obtain a hierarchical set of structures that will be utilized to construct the intermediate graph. The actual graph construction starts with the primary structure that contains at least one key point, that subdivides it into multiple substructures. We first construct the initial graph for these substructures as described above. For secondary and higher-order structures the subgraph construction slightly differs, because by definition these structures start with an element that has three possible neighbors. Setting up the subgraph of a higher order structure might cause element ambiguities because the optimal element starting this structure might be a different one than the one chosen in the parent structure. To avoid such ambiguities, we construct separate graphs for each element that is allowed to start this current structure. We recurse into child structures until we have a full set of subgraphs. In a final step, the child structures need to be merged into the graph of its parent structure. Here, we proceed as illustrated in Figure 3.4. Assume we have created separate graphs for substructure $k_1 \to S_4$, each starting with a different element $e_i$ with more than two neighbors. In the parent structure, this corresponds to an element $e_i^p$, of which we create a *virtual duplicate* $e_i^{p'}$ that consumes zero resources and retains the outgoing edges from the original. The outgoing edges of the original $e_i^p$ are replaced by new edges that connect to the front element of the substructure. Its back elements, in turn, are linked back to $e_i^{p'}$ via zero-cost edges. This procedure allows us to merge a child structure into the graph structure of its parent without causing element ambiguities. At that point, the choice of a specific element (for instance, a T-shaped part) will thus depend on all its neighbors, rather than just on the usual two. Closed curves are treated similarly. Here, we simply split them at a key point (Figure 3.5). The main difference to dead-end branches as described above, are real costs associated with the edges that close the cycle. In case if a cycle and t-junctions are present, we split and unroll the cycle at a key point and use the unrolled cycle as primary structure. Then the remaining higher order structures can be computed as described above.

## 3.4 Complexity Analysis

In the following, we analyze the complexity of our algorithm for one resource $R$. Let $n$ denote the number of elements in a database and $d$ denote the maximum number of concatenation neighbors of an element. As we described in Section 3.3.1, the algorithm performs two expansion steps, forward and backward. Each step consists of two nested *for*-loops. The outer *for*-loop is executed at most once for each node on a current level. There are at most $n$ nodes in each level since a new node on a level is created at most once for each element of the database. The inner *for*-loop is executed for each concatenation neighbor of the current outer loop node, thus at most $d$ times for each node of a level. Hence, the runtime of each step is in the order of $\mathcal{O}(dn)$. Since each step, including the initialization, is executed at most once for each level, the overall runtime of the algorithm is in the order of $\mathcal{O}(Ldn)$, where $L$ is the number of levels. By definition $L = \frac{R}{GCD}$, thus the complexity of the presented algorithm is in the order of $\mathcal{O}(\frac{R}{GCD}dn)$.

## 3.5 Comparison and Analysis

Our technique requires a bit of effort, and it may seem to the reader as if existing, possibly simpler, approaches should easily generalize to the RCKSP class of problems. We found that this is not the case—in fact, we only developed the proposed algorithm after finding that the most promising existing technique, computing a constraint shortest path, in the fashion of Lefebvre et al. [LHL10] is not capable of computing multiple optimal solutions, cannot handle complex structures and suffers from huge performance issues when multiple constraints are required and the element database is large. Zhou et al. [ZJL14] use a dynamically growing table instead of a graph. However, we found that such approaches cannot be extended to reasonably deal with our larger set of goals, namely complex structures and multiple optimal solutions.
We will use the remainder of this section to explain why we opted against using any of the existing resource constrained shortest path algorithms such as the one by Lefebvre et al. [LHL10], Zhou et al. [ZJL14] or Turner [Tur12], which we will summarize with the term constraint forward search. The "canonical" extension of the constraint forward graph search technique introduces a notion of states, which are identified by an element $e$ and a vector of currently consumed resources $\mathbf{r}^c(e)$. Each state also stores the current path cost. Starting from the source node, states with minimal cost (as opposed to minimal resource consumption) are expanded first until a state is reached that satisfies all the resource constraints. States that violate any constraints are not expanded further as it is realized in our approach. When a valid goal state is reached, the algorithm terminates and outputs the found

optimal solution. This simple algorithm works because it relies on bounding the states with the integer simplification as it is done by our method as well. However, as it can be easily seen, this simple algorithm has several flaws and it is not useful in our particular problem setting. First, this algorithm uses the cost in order to sort or rank paths which are immediately explored next. This strategy, is known to be very inefficient when used for RCSP problems in theory (see Ziegelmann [Zie04], Handler et al. [HZ80] and Skiscim et al. [SG89]). The fundamental problem of such an approach is that it lacks an explicit correlation between the total cost and the feasibility of a solution in terms of resource constraints. Therefore, the number of paths that need to be enumerated might be very high until the first optimal *and* feasible solution is found. By enumerating states by cost, the algorithm is likely to be misguided towards exploring cheap but infeasible paths first, which may heavily affect its runtime.
A further gain in performance is attained using the proposed bidirectional search. Although there are cost-oriented algorithms for bidirectional search such as the approach of Lo and Zwicker[LZ10], in order to achieve competitive performance, they require a carefully designed strategy to balance the growth of the forward and backward search trees. In addition, such approaches are not capable of delivering multiple optimal solutions. The outlined forward graph search technique would, in principle, be able to find multiple solutions by running the algorithm multiple times and suppressing edges from previous solutions. However, this implies that the guarantee of optimality is lost for all solutions except the first one. For some applications, however, the optimality of the generated solutions is of great importance; see, for example, Safonova and Hodgins [SH07]. Finally, we are not aware of any way how existing forward search techniques might be efficiently extended towards globally optimal 1.5D or cyclic structures.

# CHAPTER 4

# Example-based Road Network Generation

## 4.1 Hierarchical Road Network Generation

### 4.1.1 Motivation

Existing approaches for urban modeling, whether procedural approaches, inverse procedural approaches, or example-based approaches share one common idea: *They attempt to synthesize or reproduce visually convincing results by mimicking the style of existing real-world examples*.

Procedural approaches and inverse procedural approaches rely on a fixed or learned rule set respectively; however, in both cases, the rules or the grammar represented by them can be interpreted as an abstract representation of one style or multiple styles found in the real world. Rule-based approaches have been used for the synthesis of street networks [PM01, KMK12, FBG$^+$16, BWK14, KM07, EBP$^+$12]. The reader is referred to section 2.1, where a brief overview of existing road network generators is discussed. In order to define these rules, grammars generally exploit distinctive geometrical or structural relationships such as a grid or a radial based road patterns or grid-based façade layouts. Unfortunately, finding a viable grammar is far from being trivial. For more complex structures such as city layouts with complex structured road networks and a huge variety of individual buildings, the use of a procedural method often requires an extensive amount of manual work such as rule definition, parameter tuning, and modeling of custom 3D geometry.

In contrast to procedural approaches, captured or modeled examples in combination with custom tailored algorithms might be used for the content generation

process. Methods that re-use existing content in order to synthesize novel content are called example-based approaches. In contrast to procedural approaches, where diversity and style are encoded in rule sets, real-world exemplars encode the diversity and the style found in the real world. Example-based approaches encapsulate the knowledge how these examples might be recombined to shape novel content, either within a learned model or a custom-tailored algorithm. Such algorithms have already been proposed for the synthesis of road networks [AVB08, NGDA16a] and even the synthesis of buildings or building like structures [BA05, Mer07, MM09, LCOZ+11] several approaches can be found when reviewing the literature (see section 2.1 and section 2.3) for additional details.
Example-based approaches typically require only a small amount of user interaction, i.e., a few sketches or scribbles in combination with the selection of an example database are enough to steer the custom-tailored algorithm. This significantly reduces the user interaction with the example-based system and provides a very flat learning curve when compared to procedural approaches, where rule sets need to be defined, and a large set of parameters need to be correctly set in order to produce the desired result. Instead, all relevant information and features are extracted from the exemplars and the user input. With emerging geographical information system (GIS) repositories such as *EarthExplorer* [Ear17] or *OpenStreetMap* [Ope17a] publicly available high-quality road network data and cadastre data is available that can be used to feed example-based urban modeling techniques with a nearly infinite amount exemplars.
However, even with a significant amount of high-quality example data at hand, the creation of urban layouts remains a challenging task. The major reason is that there is no a-priori notion of style and semantics that can turn a given user input into a plausible city layout incorporating both streets and buildings. In this part of this thesis, we make the fundamental assumption that fragments within existing city layouts represent the look of cities locally. Such a fragment can be seen as a continuous region within a city similar to a district. It may contain other streets and information about the street topology, building footprints, topographic relief, and even semantic information about land use and building density. Naturally, larger districts are subdivided by roads into smaller regions that are themselves subdivided by even minor roads. At the very end of that hierarchy regions completely enclosed by roads - the city blocks - containing building footprints and dead-end roads are left over.
The approach for synthesizing a new urban layout, proposed in here, exploits the described hierarchy and includes a novel hierarchical fragment based algorithm for synthesizing street networks. The fragments are extracted from example city maps by a hierarchical decomposition of the road network in a first step. A novel road network is generated from sketched arterial streets by applying a recursive warping-based fragment insertion scheme. In this section, we will focus on the

synthesis of road networks using the mentioned fragments. A technique that synthesizes building footprint layouts and places 3D buildings models on top of them is presented in section 5.1.
Since the internal structure of fragments - i.e., the placement of streets and footprints - is preserved, a plausible style transfer from real cities to virtual cities is possible. One advantage of using fragments is that they can be annotated with additional semantic knowledge, e.g., street pattern, building density, land use or building height distribution. Such semantics can then be incorporated into the synthesis algorithm using appropriate weights to prefer specific characteristics.
Using examples from publicly available databases such as *OpenStreetMap* [Ope17a], a plausible street network can be generated in a fully automated fashion without the need for any expert knowledge. By simply choosing viable examples, that contain the streets and street patterns of different cities, their individual style can be easily transferred to a novel virtual road network.
The approach presented in here categorizes itself into works on example-based urban modeling. Our approach differs from Aliaga et al. [AVB08] by the use of closed fragments.These fragments are more flexible than the crossings used by Aliaga as their use allows to maintain the road topology when copied or warped to a target region. Thus, structures such as plazas and round-abouts are present in the synthesized results. In addition, the fragments can be placed terrain-aware as each fragment stores its topographic information where it was placed in the real world. Furthermore, semantic attributes can be propagated from lower hierarchy levels towards higher hierarchy levels and even allow the semantic placement of buildings or even road patterns according to that information. The very end of the hierarchy is able to store real-world building footprints which can be copied into the city blocks to shape an initial building footprint layout including the semantics of the transferred buildings. In contrast, Aliaga procedurally generates parcel structures and fills them by copying image fragments from real-world satellite imagery that typically have semantics attached and thus cannot be transferred to the final layout.
Yang et al. [YWVW13] uses a hierarchical set of templates similar to our fragments. Their templates are hand-made and heavily annotated in a preprocessing step. In contrast, our fragments are extracted from real-world cities. This enables access to a large diversity of different styles and to capture real-world semantics found in real-world city districts.
Nishida et al. [NGDA16a] use road patches from real road networks as templates. In contrast to our closed fragments, they focus on growing a novel road network by copying road patches in order to preserve special structures such as roundabouts and plazas. In cases were no example-based growth can be performed, they employ a procedural growing scheme similar to that of Aliaga et al. [AVB08] is performed. In a final step, buildings and additional details are procedurally syn-

thesized. The methods of Yang et al. [YWVW13] and Nishida et al. [NGDA16a] contain several improvements that are not included in the proposed approach here. However, the proposed approach was discussed by Mandt [Man11] in a rudimentary version, was presented to our project partners in 2011, and then submitted for review in 2012. At the time presented, our method could only be compared to Aliaga [AVB08] and, thus, the contribution should be related to their work.

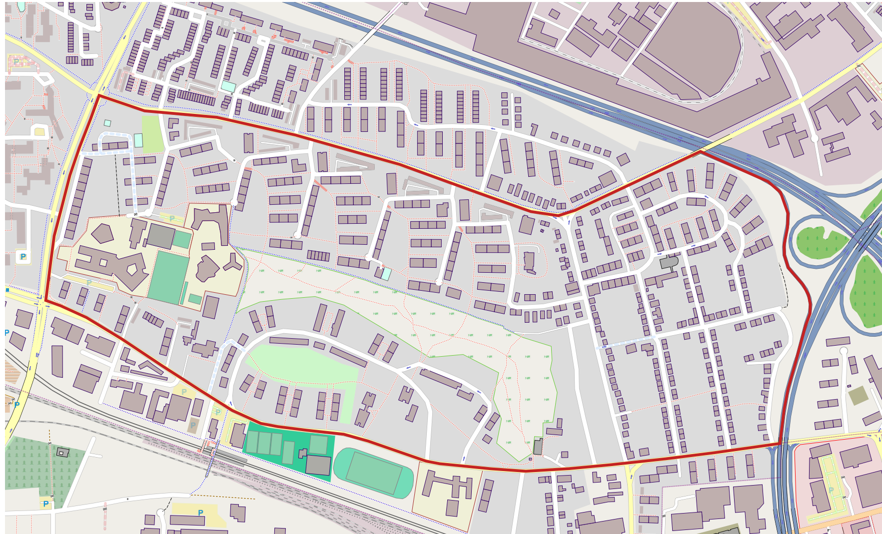


Figure 4.1: A fragment that is part of the city of Bonn taken from OpenStreetMap is shown here. Such a fragment locally represents the style of the street network and the building layouts found in an individual district of a city. The red line indicates the boundary of the fragment.

### 4.1.2 Hierarchical Fragment Construction

In this section, we discuss the preprocessing step for the decomposition of an example city map into a set of hierarchical fragments. For all examples presented in this work publicly available road networks retrieved from *OpenStreetMap* [Ope17a] are used.

**Hierarchical Example City Decomposition** The goal is to exploit the natural hierarchy of streets that is present in a multitude of real-world cities: Major streets typically enclose urban districts, each having characteristic properties related to land use, topography, street pattern, and building distribution. Minor streets subdivide these districts even further. This allows to decompose them into smaller entities and in even finer and more localized parts. Our goal is to catch these characteristics on different street levels and store them as semantic entities of cities

called fragments that will be the basic building blocks for the road network synthesis algorithm. In order to extract these fragments from given example city maps, a hierarchical representation of the streets is necessary in the first place in order to identify the fragments that live on different hierarchy levels. The concept of street hierarchies, has previously been discussed in urban planning literature such as Marshall [Mar04] and was also employed in the work of Aliaga et al. [AVB08] to annotate individual intersections with a street hierarchy level. In our setting, individual road segments will be annotated with a street level.

In order to form a street hierarchy, the streets of a given example city are inserted into a graph $G = (V, E)$ where $V$ denote the vertices, i.e., street intersections of the street network, and $E$ denote the edges, i.e., piecewise linear street segments. The hierarchical street labeling is performed from an initial set of arterial streets, i.e., important streets that define the skeleton of the street network. On the one hand, the set of arterial streets can be directly determined from the exemplar road network in the case it provides meaningful annotations or street labels. On the other hand, the labels can also be determined by computing a set of the longest streets or a set of streets that have small direction changes within $G$. From the initial arterial street set, labels for the remaining streets in $G$ are determined inspired from stream order proposed by Holton [Hol94]. Such an ordering has been used to determine a hierarchy in rivers and stream networks or the hierarchy of stems and branches in trees. The arterial streets and their segments within the street graph $G$ will be labeled with street-level 0 to provide an initial label. Branching streets of the currently processed street level will be labeled with the next higher level. However, a street network of a city is typically a graph, rather than a tree and calls for an adaption of the stream order to properly handle loops (cf. Figure 4.2). The hierarchical labeling typically results in four to five street levels.

We use the hierarchical street representation in order to extract fragments that later serve as building blocks for the synthesis step. Before, we continue explaining the method for the fragment extraction we first define the notion of a fragment $F_l$. A fragment $F_l$ of street-level $l$ is defined by a boundary $B_{F_l}$ and a set of inner/interior streets $I_{F_l}$. Streets segments located on the boundary $B_{F_l}$ of the fragment $F_l$ are labeled with street levels $L \in [0, l]$. The streets that are part of $I_{F_l}$ are by definition of street-level $l + 1$. The boundary $B_{F_l}$ is later used for the *Fragment Retrieval* step, the interior streets are used in the *Fragment Insertion* step. Both steps are explained in section 4.1.3. The computation of the regions that form fragments of a specific street level $l$ can be performed by computing minimal cycles within a subset of $G$. To be more specific, the minimal cycles are computed by only taking the street segments labeled with street-level $l \in [0, l]$ into account. The computed minimal cycles form the boundary $B_{F_l}$ of the fragments of level $l$. A fragment might enclose street segments with higher hierarchy levels than $l + 1$. However, only those street segments that have level $l + 1$ are extracted and stored in $F_l$ (cf.

Figure 4.3). In cases where a fragment is not subdivided into smaller blocks, i.e., it only contains dead-end streets, the fragment represents a city block (see Figure 4.3). The fragment is then stored as a city block, and its interior streets are stored along with it. Such fragments later produce dead-end streets in the final layout.

The extracted fragments $F_l$ are stored in a hierarchical fragment database and can be combined with other fragments of same hierarchy level into street pattern style collections (e.g. *Grid*, *Irregular*, *Suburban*, *Mediterranean*). Additionally, each fragment stores a part of the relief from the example city it was extracted from.

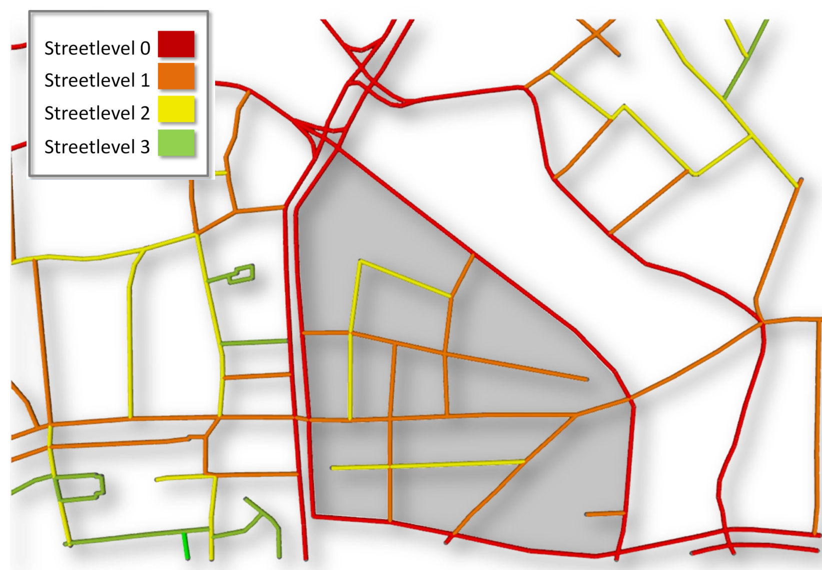



Figure 4.2: The hierarchical street representation illustrated using a part of an example city. Streets of different hierarchy level are colored differently. The gray shaded region represents the area of a fragment of level 0.

### 4.1.3 Hierarchical Synthesis of Street Networks

After describing the offline preprocessing step in Section 4.1.2 that creates the hierarchical fragments from a given exemplar city map, we proceed in this section with a detailed description of the novel hierarchical urban layout synthesis technique based on recursive fragment insertion. As our method requires user input in order to synthesize a novel street network, we start with a brief explanation of the required user input that is necessary.

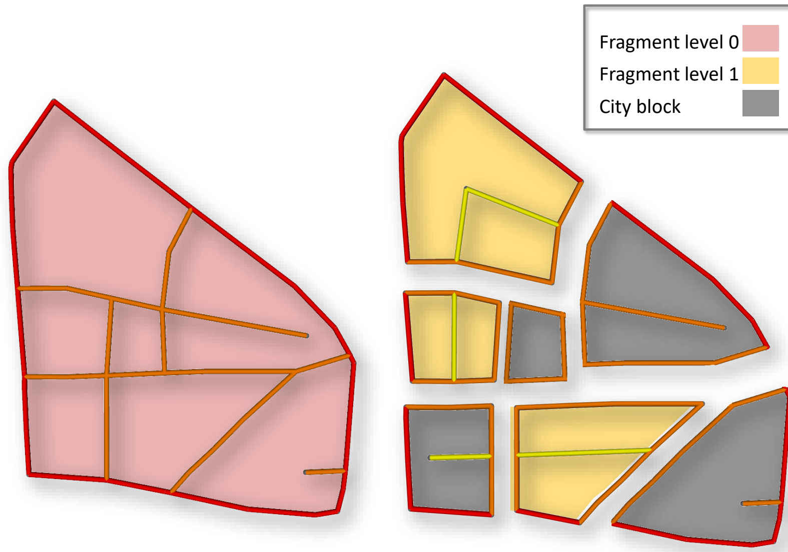



Figure 4.3: The extracted level 0 fragment (cf. gray shaded area in Figure 4.2). This fragment subdivides into a set of level 1 fragments containing streets that may subdivide these fragments even further. Regions that cannot be further subdivided will be stored as city blocks.

**Sketch Based User Input.**

The minimal input for the urban layout algorithm is at least a set of arterial streets that need to be provided by a user. The user may sketch a set of arterial roads $A = \{A_1, \ldots, A_k\}$ using the sketch interface. Here, each arterial road $A_i$ is represented as polyline $A_i = \{p_1, \ldots, p_n\}$ that is composed of $n$ points $p_i$, where each $p_i \in U \subset \mathbb{R}^2$ is located in the urban layout $U$ which is a closed subset of $\mathbb{R}^2$. The user may provide optional input such, as a relief map $R$ that contains the desired terrain topography, or attributes maps that contain information about the street pattern style, land use style or the architectural style in specific regions of the urban layout as images. A representative user input that shows arterial street sketches, as well as the topographic relief map, is shown in Figure 4.7.

**Street Network Synthesis Algorithm.**

The street network synthesis technique uses the $k$ sketched arterial streets $A$ and constructs an initial street graph $G = (V, E)$ where each of the edges is labeled with street-level 0. The initial street graph $G$ is used to compute a set of empty fragments $F_0^q$ that are used for the synthesis of the next higher street level 1. The

boundary $B_{F_0^q}$ of the fragments $F_0^q$ is determined by the computation of minimal cycles taking only street segments that have street level $l = 0$ assigned into account.

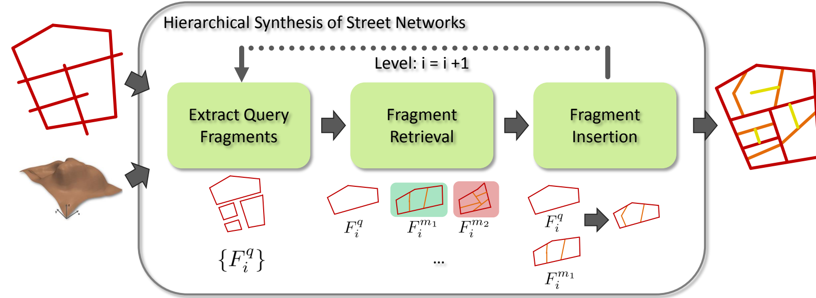



Figure 4.4: Our hierarchical street network synthesis pipeline

For the general case of street-level $l$ the boundary of the fragments $F_l^q$ are the minimal cycles of $G$ taking only street segments labeled with $l \in [0, l]$ into account. The following steps are performed consecutively for each hierarchy level $l$ starting from level $l = 0$ as outlined in Function 2.
For each empty fragment $F_l^q$ of hierarchy level $l$ extracted from the street graph $G$, a fragment with similar characteristics, i.e., similar boundary shape and topography, is retrieved from the *fragment database*. In case if any style maps are provided, only candidates matching these specific criteria (e.g., street pattern, land use) are considered for the fragment retrieval step.
The most similar candidate fragment $F_l^m$ retrieved is incorporated for the insertion into the empty fragment $F_l^q$. Here, the most similar fragment is determined by the total matching cost that measures the deviation in shape and the deviation in elevation (see paragraph *Fragment Retrieval* below for a detailed explanation). The local street topology of the interior streets $I_{F_l^m}$ and thus the street pattern located within the candidate fragments is preserved by warping the interior streets of the retrieved fragment $F_l^m$ into $F_l^q$. The same warping is performed for the fragment topography. As the local topography of the fragments from real-world cities differs in their elevation level a smooth transition, especially at the fragment boundaries, needs to be ensured. In here, we ensure this smooth height transitions at the fragment boundary of $F_l^q$, by merging the relief of $F_l^m$ using *Poisson Image Interpolation* introduced by Perez et al. [PGB03].
The algorithm stops if no further fragments of hierarchy level $l$ can be extracted from the street graph $G$. After the algorithm has terminated a novel virtual road network has been constructed that might be further enriched with building footprints and 3D building models. Such a step is discussed in chapter 5. Additionally, it proposes a re-synthesis of a building footprint layout for the case when the

```
 1: level ← 0
 2: frags ← GetFragments(level)
 3: l ← Length(frags)
 4: while l > 0 do
 5:    for i = 1 → l do
 6:       style ← DetermineFragmentStyle(frags[i])
 7:       f ← RetrieveMatchingFragment(frags[i], style)
 8:       InsertFragment(f)
 9:       PropagateHeightDistribution(f)
10:    end for
11:    level ← level + 1
12:    frags ← GetFragments(level)
13:    l ← Length(frags)
14: end while
```

Function 2: The different steps performed by the hierarchical street network synthesis algorithm.

shape difference between the query fragment and the matched candidate fragment is large. For now, we continue with the retrieval of the candidate fragments.

### Fragment Retrieval

In order to identify viable candidate fragments from the database we adopt the *Shape Context* descriptor initially introduced by Belongie et al. [BMM00] and extend the induced shape similarity measure by a cost term accounting for similarity in height. Incorporating height is necessary because fragment characteristics, i.e., street topology, building distribution and land use in flat urban areas commonly differ from those of hilly regions. A classic example is that a serpentine curve typically shows a very different style when compared to ordinary road courses.
Given a set of points $p_i$ that are uniformly sampled from the outline of a 2D shape, the *Shape Context* at a point $p_i$ characterizes the local neighborhood of $p_i$ in terms of a histogram over the distances from $p_i$ to neighboring outline points. Let the dissimilarity of two Shape Contexts of points $p_i$ and $p_j$ be denoted by $C^S_{ij}$. To determine the distance between two shapes $P = (p_1, .., p_n)$ and $Q = (q_1, ..., q_n)$ according to the sets of underlying Shape Contexts, a (cyclic) permutation $Pi(Q) = (q_pi(1), ..., q_pi(n))$ is computed, such that the resulting matching costs $C(P,Q) = \sum_{i=1}^{n} C^S_{i,\sigma(i)}$ are minimized. The shape context is configured to be rotation invariant, i.e., the log-polar histogram is aligned with the local tangent computed at each sample point. In addition, we need the matching procedure to respect the scale of the fragments. This is achieved by using the inter-point dis-

tance inside the log-polar histogram without normalization by the mean distances between all sample points. To incorporate additional fragment characteristics such as topography or land use, we extend the original approach by additional matching cost terms, such that the resulting land use, building density or building height reads:

$$c\,(P,Q) = \sum_{i=1}^{n} \sum_{j=1}^{m} \gamma_j C^j_{i,\sigma(i)} \tag{4.1}$$

where $\gamma_j \in [0,1]$ and $\sum \gamma_j = 1$ holds. $C^j$ denotes the matching costs for the particular fragment characteristic and $\gamma_j$ governs the balance between the individual characteristics. In our setting, we incorporate shape and size as well as the relief. Therefore, equation 4.1 simplifies to

$$c\,(P,Q) = \sum_{i=1}^{n} \gamma C^S_{i,\sigma(i)} + (1-\gamma) C^E_{i,\sigma(i)} \tag{4.2}$$

where $C^S_{ij}$ denotes the shape term and $C^E_{ij}$ denotes the elevation term, taking the relief of the fragments into account. Let $F_m$ be a candidate fragment, that matches the characteristics, shape and elevation, of the query fragment $F_q$. Then the costs for $C^S_{ij}$ are computed as described by Belongie et al. [BMM00] utilizing uniform samples on the fragment boundaries (see Figure 4.5).
To determine the costs for matching the different elevation levels $C^E_{ij}$, we first compute the height difference between boundary points $p_i$ and the centers of the Shape Context bins (see Figure 4.5). Let $E_{pi}$ denote the height at $p_i$ and let $E^t_{bin}$ denote the height at the center of Shape Context bin $t, t = 1, \ldots v$. We store the resulting slopes $sl_{it} \frac{E^t_{bin} - E_{pi}}{l}$ in a descriptor $SL(i) = (sl_{i1}, \ldots, sl_{iv})$. The elevation based costs for matching points $p_i$ and $p_j$ can now be defined as

$$C^E_{ij} := \frac{d(SL(i), SL(j))}{2\sqrt{(v)}} \tag{4.3}$$

where $d(.)$ denotes the Euclidean distance. Note that as we mostly consider urban terrain, where the local slope is not larger than 1, which ensures that $C^E_{ij} \in [0,1]$. Using this differential height descriptor makes the cost independent of the actual heights, and better reflects the fact that the slope is affecting streets more than absolute heights.
The similarity measure for a pair of fragments is finally derived by minimizing the total matching cost of all pair permutations $C_{ij}$. This can be achieved by computing the least cost path through the cost matrix $C$ using *Dynamic Time Warping* (DTW) [SC78]. Because conducting DTW between $F^q_l$ and all possible candidate fragments $F^m_l$ would be too costly to compute, we first prune inappropriate

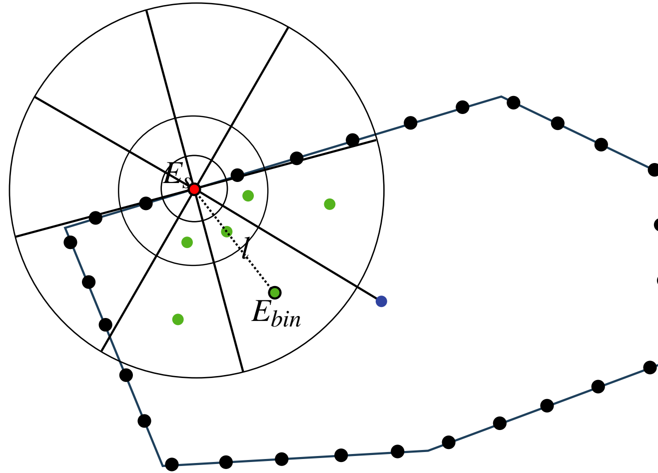



Figure 4.5: Example fragment $F_l$ uniformly sampled along its boundary. The log-polar histogram represents the Shape Context for a single sample point $p_i$ (red) of $F_l$. The histogram is aligned at the 0° bin to the center of gravity (blue dot) of $F_l$. The green dots illustrate the sample points $E_{bin}$, used to compute the slope between $p_i$ and $E_{bin}$ in the elevation term ${C^E}_{ij}$

fragments of the current hierarchy level $l$, by clustering their size, aspect ratio, and relative elevation changes. In our experiments, we use the first $c = 10$ candidates in order to compute the similarity measure between $F_i^q$ and the $F_i^m$'s using *Dynamic Time Warping*.

### Fragment Insertion

Once the most similar fragment has been retrieved from the database it is inserted into the synthesized street network. Naturally, position and orientation of the closest match $F_l^m$ differs from that of $F_l^q$. Therefore, $F_l^q$ and $F_l^m$ are rigidly aligned using the correspondences between sample points on the boundaries of $F_l^m$ and $F_l^q$. These correspondences can be easily extracted from the least cost path through the cost matrix $C$ that was previously computed within the fragment retrieval step by the Dynamic Time Warping algorithm. During the insertion of a matched candidate fragment, we would like to preserve the street pattern and the street topology. Thus, we are able to transfer round-abouts, and intersection configurations as they are present within the real world. Simply copying the inner streets $I_{F_l}$ of the matched fragment $F_l^m$ is inappropriate, because street connections towards the boundaries might not be established in $F_i^q$. In addition, the street structure might get destroyed, because parts of the street segments would be cut off. This results from individual streets located inside $F_l^m$ might be located

outside $F_l^q$ due to boundary shape differences. Instead of copying the interior streets of $F_l^m$, we warp them into $F_l^m$ using *Generalized Barycentric Coordinates* (GBC) introduced by Hormann [HF06]. In detail, we compute the barycentric coordinates for each intersection located within $F_l^m$ reusing the points that were sampled along the fragment boundary and were used for computing the *Shape Context* descriptors from the fragment retrieval step. The final positions of the intersections located inside $F_l^q$ are computed using the sample points of $F_l^q$ and the point-to-point correspondences derived from the least cost path through the cost matrix $C$. The same warping approach is employed to transfer the relief of the fragment $F_l^m$ into the height-field of the urban layout. Generally, the relief of the matched fragment $F_l^m$ is located on a different elevation layer. Just warping the relief is not enough, because unpleasant discontinuities would be visible in the final height-field. Smooth transitions between fragment boundaries are ensured by propagating the gradient field of the relief into the height-field by employing *Poisson Image Editing* introduced by Perez et al. [PGB03]. The main advantage of the warping approach is that the local topology of the street segments of $F_l^m$ and therefore the street pattern is preserved. Another advantage is that the relief of the matched fragment is smoothly transferred into the current state of the topography map.

### 4.1.4 Results

The effectiveness of the road network generation algorithms was evaluated for a multitude of different challenging use cases. These include pure street network generation, to the generation of complex urban layouts that show a variety of different street patterns.

The ability to retrieve and identify similar fragments is of fundamental importance for successfully synthesizing plausible street and building footprint layouts. The first test we conducted is the reproduction of a small part of the city of Bonn shown in Figure 4.1.

The result of this experiment is illustrated in Figure 4.6. It shows that the fragment retrieval is able to reproduce the look of a given city district from the respective arterial streets and a fragment database containing fragments of a complete city. For this example, we used a database containing the fragments of the city of Bonn. The sketched boundary highlighted in red and the underlying terrain information were expressive enough that our method is able to re-synthesize the street patterns found in the district shown in Figure 4.1. In addition, we synthesized building footprint arrangements using the technique showcased in section 5.1.

Figure 4.7 and 4.8 show a synthesized road network for which only suburban regions extracted from the city of Boston were used. The curvy street layout commonly found in suburban regions is faithfully reproduced. However, we can notice

Figure 4.6: The reproduced look of a district of a real-world example city: Red sketched arterial streets serve as input. The recursive fragment synthesis algorithm faithfully reproduces the look of the region shown in Figure 4.1. Notice that even the style of building footprint layout is similar when compared to the original layout. The 3D building models were placed using the technique presented in section 5.1

that in specific regions the streets are unduly warped. This is a direct consequence of large shape differences between the query and candidate fragments. Here, the used warping technique introduces heavy distortions that locally produce road courses that look implausible. Another property of rural regions is a sparse distribution of buildings within the city blocks. We used the technique proposed in section 5.1 in order to transfer the building footprint layout from suburban fragments extracted from the city of Boston to the synthesized road network. The result shows that our approach is able to reproduce the sparse building footprint distributions, that are commonly found in suburban regions, within the virtual city layout.

As discussed in Section 4.1.3 there is a relationship between the topography and the street patterns that occur in different regions. The presence of steeper slopes within the terrain topography should affect the synthesized street pattern in such areas. A synthesized result using our approach is depicted in Figure 4.9, were we used fragments from Kopenhagen (DK) and San Fransico (USA, CA). This result illustrates that our method is able to adapt the style of the street pattern according to the actual terrain. Hence, this leads to more curvy street pattern appearance in hilly areas, while in regions dominated by flat ground, irregular streets patterns from the fragment database of Kopenhagen are selected.

One of the strengths of our approach is that style of both streets and buildings can be easily controlled by user defined maps. Figure 4.10 shows the result of mixing a

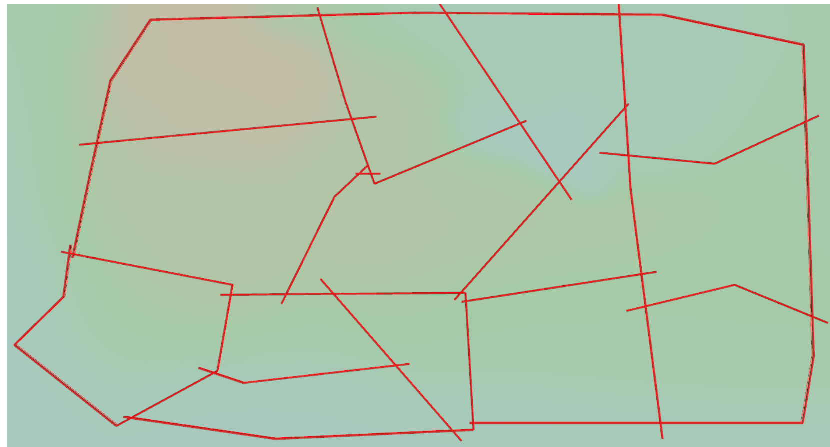


Figure 4.7: A sketched arterial road network provided by the user

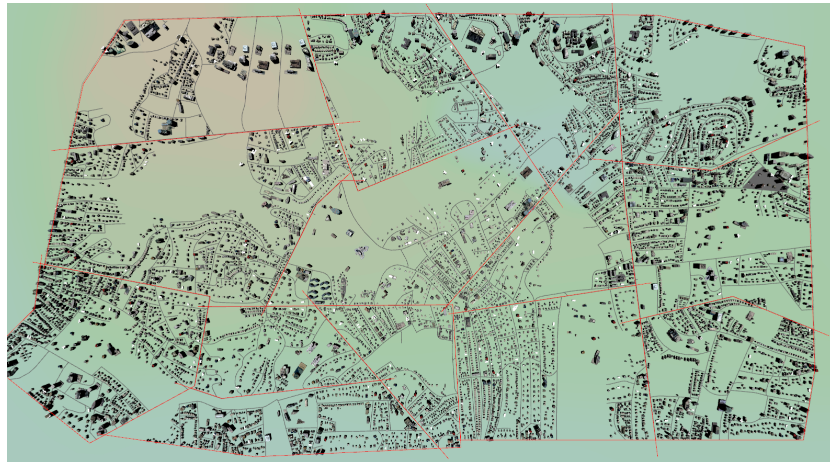


Figure 4.8: A synthesized urban layout that was generated using fragments extracted from the suburban area from the city of *Boston NY*. Notice the style of the distinctive style of the street network mostly occurring in suburban areas of US cities is preserved.

grid-based street layout, typical for North American metropolises, with irregular street patterns commonly found in European cities. Comparing statistics based on fragment aspect ratio and fragment area for both, example and synthesized road networks clearly indicates that style transfer is effective (see Figure 4.11 and 4.12).

Figure 4.9: Topography Example: The example was generated using a sample from San Francisco (Mount Davidson Park, Mount Sutro) and Kopenhagen. In the area enclosed by the red polygon, fragments from San Francisco were chosen because of the similar topography distribution. Notice that in this hilly area, the street pattern style differs drastically from the street pattern style in other parts in the urban layout.

### 4.1.5 Analysis and Comparison

Existing methods are also able to produce detailed urban layouts that consist of complex street networks and building footprint layouts. The method of Yang et al. [YWVW13] shares the idea of using hierarchical building blocks. However, in contrast to our real-world fragments, they rely on manually designed templates that capture common subdivision patterns found in suburban regions. The number of annotations necessary to produce such templates might be cumbersome when other street patterns should be integrated. Therefore, an automatic approach that extracts templates from real-world data as we did enables access to a large variety of street patterns. Compared to Aliaga et al. [AVB08] the use of templates allows the preservation of topology structures such as roundabouts and plazas as it was confirmed by the more recent work of Nishida et al.[NGDA16a]. The use of closed templates enables the propagation of semantics and urban structures such as footprints to different hierarchy levels, that is more difficult to achieve when 'open' templates are used as done by Nishida.

The integration of the topographic information allows the integration of additional semantics, namely, the constraint placement of slope specific road structures (see Figure 4.9). However, the placement strongly depends on the variety of different

Figure 4.10: The resulting street network was synthesized using fragments from the city of Kopenhagen (irregular street pattern) and from Phoenix (grid) pattern. The street network contains 883 city blocks, and the whole street length is 532 km.

templates that are present within the database. Having too few templates, directly result in recognizable distortions introduced by warping, and repeated insertion of one or multiple fragments. When the number of templates is large, the probability of finding a template with a moderate shape difference increases. Thus, the distortions that might affect the street layout within a fragment might be neglected.

### 4.1.6 Limitations

We identified several limitations of the proposed road network synthesis technique, that will be discussed in the following. The current approach performs the recursive insertion strategy for each fragment individually. This strategy, however, limits the creation of street connections between neighboring fragments. These inter-fragment connections are especially necessary for improved traffic flow and efficient movement within the street network. In the presented approach, these connections are established by luck, and no guarantee can be given that larger structures will be formed on higher hierarchy levels.

The number of fragments that can be extracted on the different levels strongly depends on arterial streets that are used to compute the hierarchical street labeling. Typically, only a few fragments can be extracted on lower hierarchy levels when

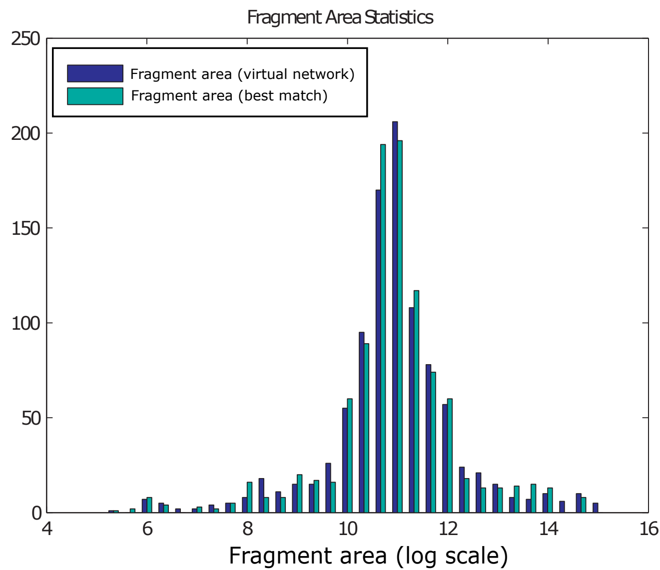



Figure 4.11: The illustrated histogram shows the distribution of the fragment area within the synthesized road network and the fragments that chosen as the best match during the fragment retrieval.

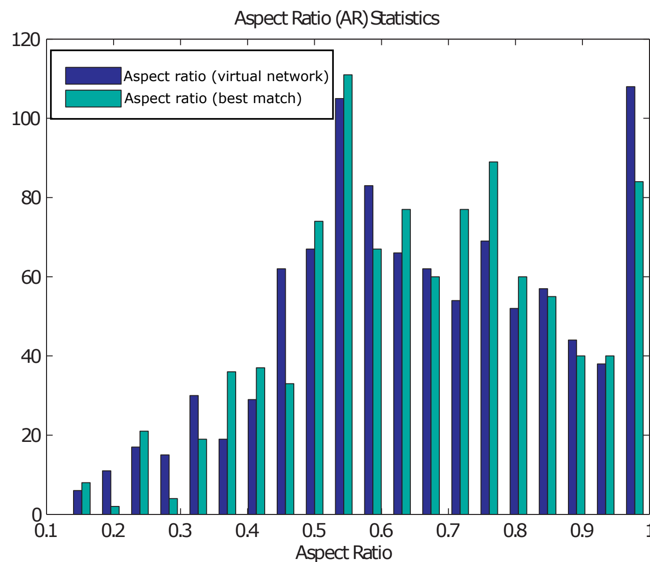



Figure 4.12: The illustrated histogram shows the distribution of the fragment aspect ratio within the synthesized road network and the fragments that were chosen as the best match during the fragment retrieval.

compared to the number of fragments that can be computed on higher hierarchy levels. For the synthesized road network, this will result on a recognizable repeated subdivision pattern on lower hierarchy levels, because the same fragment might be multiple times the best match. This effect can be alleviated by choosing a random fragment among the *n* best matches; however, in combination with the risk of an increased shape difference and thus unpleasant warping artifacts.
The number of fragments available within the database also has a reasonable effect on the quality of the final layout. When a small number of fragments is available, the probability that a suitable fragment with only a small shape different is available is typically very small. Thus, the shape difference between the query fragment and the best-matched fragment will be large. When the streets are transferred using the generalized barycentric coordinates, the interior streets are extensively warped especially at the boundary. This effect intensifies when additional attributes such as topography, land use, or other custom types should be incorporated into the street network generation process. In rare cases, warped intersections might be located outside the target domain. This might happen when the target domain has an extremely concave shape. Such cases need to be detected and instead of inserting a fragment with a completely different shape. A fall back that procedurally splits the domain might help to overcome such defects.

## 4.2 Road Network Generation with Generative Adversarial Networks (GANs)

In this chapter, a novel example-based approach for road network generation using a recently introduced deep learning technique is described. The content is based on the peer-reviewed publication

> Stefan Hartmann, Michael Weinmann, Raoul Wessel, and Reinhard Klein. Streetgan: Towards road network synthesis with generative adversarial networks. In *International Conference on Computer Graphics, Visualization and Computer Vision*, June 2017.

This chapter contains text parts of the mentioned publication that were copied without further modification. These text parts are highlighted in gray.

### 4.2.1 Motivation

Existing road network generation algorithms use either procedural approaches [PM01, BWK14, FBG$^+$16] or example-based approaches [AVB08, YWVW13, NGDA16a]. While the first rely on rules or re-writing systems, example-based approaches use custom-tailored algorithms that reshuffle, recombine, and bend the

content in order to statistically and perceptually match the style encoded within the examples. When such algorithms are used for generating novel content, the diversity and the realism strongly depends on the expressiveness of the rule sets or the variations found in the exemplars.
With the explosive dissemination of deep learning techniques recently, generative neural networks have shown tremendous potential for the automatic content generation. They have been used for texture synthesis [GEB15, LW16] or 3d shape and model synthesis [NGDA$^+$16b, HKYM16, YARK15]. Especially with the invention of generative adversarial networks (GANs) [GPAM$^+$14] a novel technique has been proposed, that is able to implicitly learn the data-generating distribution that was used to produce a set of exemplars.
In this chapter of the work at hand a novel example-based approach for road network generation that leverages the potential of generative adversarial networks (GANs) is proposed. We believe that GANs are particularly well-suited for the generation of novel content as they tend to learn the data-generating distribution as stated above. Such a technique would provide different advantages when compared to existing road network generation algorithms. First, it would be possible to overcome the manual definition of rules that encode specific street patterns. Second, tedious parameter tuning that is necessary to produce the desired output would be superfluous. Third, the use of such a technique used in combination with a custom-tailored algorithm might boost the variation of the generated results, because novel templates can be generated in order to augment real templates with synthesized ones. To the best of our knowledge, no other method for road network generation using a GAN approach has been published so far.
Before, we state our approach we, briefly review existing approaches for content generation that have recently proposed and that leverage deep learning techniques. Yumer et al. [YARK15], enables intuitive exploration of high dimensional parameter spaces. The high dimensional parameters of procedural models are categorized into shape categories and then compressed using an encoding-decoding architecture. Novel shapes can be generated by feeding a low-dimensional seed vector into the decoder which faithfully reproduces a parameter vector that can be used to generate a shape of a specific category. Novel shapes can even be generated by interpolating the low-dimensional seed vector, allowing the intuitive exploration of the parameter space. Huang et al. [HKYM16] synthesize procedural shapes from abstract hand-drawn sketches of objects. In their approach, they train a neural network to learn a map between sketches and procedural model parameters from a large set of synthetic line drawings. Novel shapes can be synthesized by feeding a hand-drawn sketch to the network, which regresses the parameters of a procedural model that closely matches the shape depicted within the user sketch. Nishida et al. [NGDA$^+$16b] follow the same line for the interactive design of procedural building models. They train neural networks to classify sketches,

drawn by a user, into categories of architectural elements, i.e., building mass, window, roof. In addition, they train individual neural networks that learn mappings from individual sketches to corresponding parameter sets of procedural models that produce a specific category of architectural elements. The user interactively sketches architectural elements to construct a novel virtual building, while their system first classifies the category of the drawn architectural element and subsequently regresses the parameters of the corresponding rule set.
Ritchie et al. [RTHG16] focus on controlling procedural algorithms using neural networks. In particular, the neural network manipulates the next steps of the algorithm based on the content generated so far. Apart from the approaches that require the existence of procedural algorithms, pure image-based algorithms have been investigated for the controlled content generation. Isola et al. [IZZE16] investigate GANs for transfer learning, i.e., they learn a mapping from one object representation into another such as *urban map to aerial image* or *sketch to image*. In other words, the output of the neural network is produced by providing a condition as input. The theory for such conditional generative adversarial networks (CGANs) can be found in Mirza et al. [MO14]. A significant advantage of CGANs is that they are able to learn such a mapping without manual feature engineering. When GANs are used for image generation, the size of the produced images is typically constraint to a specific output resolution, because at a certain point within the network topology an inner product layer is used [GPAM+14, RMC15]. In order to overcome this drawback, additional convolutional blocks can be added to the network including layers to upsample the image resolution. However, this calls for retraining of the network to learn the weights of the added filter banks. To avoid this heavy computational step, it would be necessary to eliminate the inner product layer and only use convolutional layers. Such 'fully' convolutional generative adversarial networks were recently introduced by Jetchev et al. [JBU16] in the context of texture synthesis. With their approach, they are able to synthesize images of arbitrary size. In contrast to previous approaches where a $d$-dimensional seed was used to generate an image, Jetchev et al. use $d$-dimensional vectors that are spatially arranged and thus represents a seed that lives in $R^{n \times n \times d}$, where $n$ denotes the spatial resolution. As the seed vectors are spatially arranged, their technique is called *Spatial GAN (SGAN)* and can be seen as an unrolled GAN technique. In order to evaluate GANs for the synthesis of road networks, it is crucial to synthesize road network patches of a varying scale. Thus, a technique such as presented in Jetchev et al. [JBU16] seems to be especially well-suited for the envisioned task. Therefore, we use it as the basis for our example-based road network synthesis approach.
The previously discussed technique uses images for the training step. So far, all modern deep learning framework rely on images served as input, especially when convolutions are part of the network topology. Approaches that directly work on

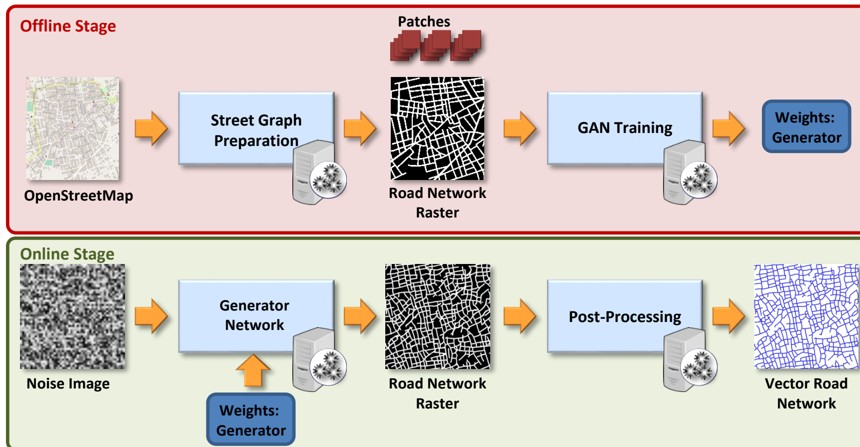



Figure 4.13: Our system is composed of two components. In an offline step, a road network patch taken from a real-world city is rasterized into an image. The rasterized road network is used to train a GAN, and the generator weights are stored. In an online step, the trained model, i.e., the generator weights, is used to synthesize road network variations from seeds that live in $R^{n \times n \times d}$ and are sampled from a simple distribution. A graph is extracted from the produced image ready to use in GIS applications.

geometry [BBL+17] or graphs [SSD+17] have been proposed recently. Because of this, these methods are still at an early stage; therefore, we stitch to the well-established image-based neural networks. In order to use street network graphs in combination with generative adversarial networks (GANs) we designed a pre- and post-processing step to successfully employ this learning technique. The proposed content generation algorithm consists of three major components that are presented in the following. The first component prepares and converts an input road network into a binary image, where the pixel intensities encode the presence or absence of roads. The second component trains a generative adversarial network (GAN) [GPAM+14] on image patches extracted from the prepared road network image. The third step utilizes the GAN to synthesize arbitrary sized images that contain a rasterized road network. In order to use the produced road network encoded in the image in GIS applications such as CityEngine [Esr17], we extract the road graph and post-process it in a final step. The results that are shown in Section 4.2.5 illustrate that our approach is able to synthesize road networks that are visually similar when compared to the original road network. Moreover, they also faithfully reproduce road network properties like city block

area, compactness and city block aspect ratio (see Section 4.2.5). A detailed statistical evaluation of the synthesized results shows that the major characteristics and the style, present within the original network are maintained.

### 4.2.2 Method Overview

Digitized road networks are typically available in a vector-based representation, i.e., roads are represented by a set of piecewise line segments. In order to successfully apply a GAN to street network data, we developed three components to approach this task (see Figure 4.13). In an offline step, a sample map from OSM is transferred into the image domain using rasterization (see Section 4.2.4). The produced image encodes the presence or absence of roads using pixel intensities, i.e. we currently do not incorporate information about the road hierarchy into the rasterized image. From such an image random patches of size $w \times h$, where $w$ denotes the width of the patch and $h$ the height of the patch respectively, are extracted, that contain parts of the original road network, and serve as training set for the GAN training (see Section 4.2.4). After training, novel road network patches can be generated from $d$-dimensional vectors that are arranged spatially into a volume $z$. The values of the individual $d$-dimensional vectors at each spatial position are drawn from a simple distribution $p_z(z)$. The generator component of the GANs takes an input such $z$'s and transforms them into gray-scale images $x \in \mathbb{R}^{w \times h \times 1}$, where $w$ and $h$ denotes the width and height of the produced image. The produced images contain a road network that is encoded by pixel intensities. The images are used in a post-processing step in order to extract a road graph (see Section 4.2.4). We use the reconstructed graph representations of the street network to conduct an in-depth evaluation of the synthesized networks. Our evaluation shows that the generated road networks share the characteristics and the style, present within the original network. Moreover, it also shows that the GAN faithfully reproduces road network properties like city block area, city block compactness, and city block aspect ratio (see Section 4.2.5).

### 4.2.3 Review of Generative Adversarial Networks

Before outlining our approach in Section 4.2.4, we provide a brief overview of generative adversarial networks (GANs) that we apply to generate road networks. GANs are a technique to learn a generative model based on concepts from game theory. The key ingredients of GANs are given by two players, a generator $G$ and a discriminator $D$. The generator is a function $G_{\theta^G}(z) : \mathbb{R}^d \to \mathbb{R}^{w \times h \times c}$ that takes as input a vector $z \in \mathbb{R}^d$ sampled from a $d$-dimensional prior distribution $p_z(z)$ such as a uniform distribution and uses the parameters $\theta^{(G)}$ to transform it into a sample image $x'$. The fabricated sample $x' = G(z)$ is an image $x' \in \mathbb{R}^{h \times w \times c}$, where

$w$ and $h$ denote its width and its height and $c$ denotes its channels. In contrast, the discriminator $D$ is a function $D_{\theta^D}(x): \mathbb{R}^{w \times h \times c} \to \mathbb{R}$ that takes as input either an image patch $x$ from the training set or a fabricated image $x'$, and uses its parameters $\theta^{(D)}$ to produce a scalar that represents the probability that the investigated sample is an example $x$ from training set, instead of a fabrication $x'$ produced by $G$. The discriminator cost is accordingly given by

$$\begin{aligned} J^{(D)}(\theta^{(D)}, \theta^{(G)}) = &-\frac{1}{2}\mathbb{E}_{x \sim p_{data}(x)} \log(D(x)) \\ &-\frac{1}{2}\mathbb{E}_{z \sim p_z(z)} \log(1 - D(x')) \end{aligned}$$

which is the standard cross-entropy for a binary classification problem. The discriminator tries to minimize $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ while it controls only the parameters $\theta^{(D)}$, however, it also depends on the parameters $\theta^{(G)}$ of the generator.
The term, $\mathbb{E}_{x \sim p_{data}(x)}[\log(D(x)]$, measures the skill of $D$ to distinguish fabricated samples $x'$ from real ones $x$ that are produced by the data-generating distribution $p_{data}$. In contrast, $\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(x')]$ measures the skill of $G$ to fabricate examples, that are misclassified by $D$ and thus considered as real examples. In the previous terms, $\mathbb{E}$ represents the expectation value of the $log$-probability, which in practice boils down to the arithmetic mean of the $log$-probabilities computed using the samples of the current training iteration. The cost function of $G$ is given by $J^{(G)}(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)})$ and its goal is to maximize $D$'s error on the fabricated examples $x'$. As both cost functions follow different goals and compete against each other, the underlying problem is described as a game between the two players [GBC16, Goo16]. One strategy to solve the described problem is in terms of a zero-sum game also called *minimax* game. The game is accordingly described by the objective

$$\arg\min_{\theta^{(G)}} \max_{\theta^{(D)}} \; -J^{(D)}(\theta^{(D)}, \theta^{(G)})$$

where $-J^{(D)}(\theta^{(D)}, \theta^{(G)})$ represents the discriminator's pay-off. The overall goal of such a game is to minimize the possible loss for a worst case maximum loss. In particular, this is realized by performing a minimization in an outer loop, while performing a maximization in an inner loop. We refer the reader to a recent tutorial by Goodfellow [Goo16] for additional details.
In practice, $G$ and $D$ are represented as neural networks and training them is similar to finding the *Nash* equilibrium of the described *minimax* game played by $G$ and $D$. A *Nash* equilibrium in such a context can be described as a parameter state $(\theta^{(D)}, \theta^{(G)})$ that is a local minimum of $J^{(D)}$ and a local minimum of $J^{(G)}$. In

order to keep the problem tractable, $G$ and $D$ are trained in an alternating fashion instead of using the nested loops as described above. Furthermore, $G$'s cost function $J^{(G)}(\theta^{(D)}, \theta^{(G)})$ is changed to $-\frac{1}{2}\mathbb{E}_{z\sim p_z(z)} \log(D(x'))$. The term in the original cost function $-\frac{1}{2}\mathbb{E}_{z\sim p_z(z)} \log(1-D(x'))$ would lead to vanishing gradients during the training, when $D$ successfully rejects examples fabricated by $G$. For early analysis of the vanishing gradient problem conducted by Hochreiter [Hoc91], we refer the reader to his thesis. Instead of previously minimizing the *log*-probability that the sample $x'$ is classified as fabricated, the new goal of the generator $G$ is now to maximize the *log*-probability that $D$ performs a wrong classification. As noted in [Goo16], that change enables both $G$ and $D$ to produce strong gradients during the final training.
For the modified game and its training, this particularly means that in one iteration $D$ is trained, while in the next iteration $G$ is trained. As we search a local minimum for $D$ and $G$, the parameters of the current component are updated in each iteration using stochastic gradient descent. We use a different update technique for the gradient that is called ADAM[KB14]. The ADAM Optimizer is typically used for training GANs as it tends to perform more stable optimization and accelerates convergence. When $G$ is trained, its parameters are tuned towards the production of samples $x'$ that are indistinguishable from the real training data and thus to fool the discriminator $D$. In contrast, when $D$ is trained its parameters are tuned to improve $D$'s skill to discriminate the fabricated samples $x'$ from the real samples $x$. For additional details about the theoretic background, we refer the interested reader to the recent works and tutorial about GANs [Goo16, JBU16].
So far, when the GAN is trained using neural networks, no well-founded theory about the determination of the success of training procedure of a GAN can be found in the literature. Therefore, it is necessary to check generated samples visually and to capture the weights $\theta^G$ of $G$ that fabricate visually pleasing outputs. Note, there is no need to capture $\theta^D$ because after the training $D$ can be omitted and only the generator $G$ is necessary to produce new samples [GBC16, Goo16, JBU16].

### 4.2.4 Towards Neurally Guided Road Network Synthesis

#### Road Network Preparation

We use publicly available community mapping data from OpenStreetMap (OSM) [Ope17a] in which road networks are represented as piecewise-linear polylines, that are attached a *highway* label in order to distinguish them from other structures like buildings, rivers, and parks. Each road is assigned a specific *highway* type representing its category. For all our examples, we extract roads from the OSM dataset that have one of the following *highway*-categories assigned: **mo-**

**torway, primary, secondary, tertiary, residential, living_street, pedestrian**. The raw road network extracted from OSM is represented as vector data in geo-coordinates. As well-established CNN pipelines require images as input, we transform the road network into a raster representation, in a first step. We start with projecting the geo-coordinates to WGS84, which is a well-established coordinate projection that transforms geo-coordinates given in *Latitude/Longitude* to meters. Next, we scale the road network that each pixel in the rasterized representation represents an area of $3 \times 3$ meters. Finally, we raster the road segments as lines with a width of 15 pixels using the line rasterization routine of OpenCV [Bra00] to produce a binary image. During this process we drop the highway label and merge all roads into the same category. The main reason for this step is to evaluate if the structure found within a road network can be learned at all. Within such an image, white pixels correspond to the presence of roads while black pixels represent the absence of roads. Please note that we inverted the colors in the Figures shown within the current chapter.

### GAN Training Procedure

We train the GAN on image patches with a fixed size of $n \times n$ pixels that are extracted from the image containing the rasterized representation of the road network. In order to provide a suitable large training set and to enable the network to capture the local statistics well, we perform the training on images patches with a size of $n \times n = 321 \times 321$ instead of using a single image. The GAN training is performed in an alternating fashion as it is proposed by Goodfellow [GPAM$^+$14]. This means that $G$ and $D$ are trained consecutively within each iteration. In the step when $G$ is trained, it takes as input a set $Z = \{z_0, \ldots, z_k\}$. As described in Jetchev et al. [JBU16], that serves as basis for our training, each $z_i$ is a volume $z_i \in \mathbb{R}^{m \times m \times d}$ where at each spatial location a $d$-dimensional vector drawn from $p_z$, typically a uniform distribution providing values $d_i \in [-1, 1]$, is inserted. The $z_i$ is then transformed by $G$ into a gray-scale image patches $x'_i = G(z_i)$ of size $x'_i \in \mathbb{R}^{n \times n \times 1}$. When $D$ is trained, a set $X = \{x_0, \ldots, x_k\}$ of $k$ image patches $x_i \in \mathbb{R}^{n \times n \times 1}$ and the set $X' = \{x'_0, \ldots, x'_k\}$ fabricated by $G$ serves as input. The samples $x \in X$ are extracted from random locations inside the training image. We refer the reader to the work by Jetchev et al. [JBU16] for additional details about the training procedure. Please note that we use only a single image, that provides patches for the training procedure, but using multiple different images would be possible and would increase the examples present in the overall training set. But although we use only a single image for all our example, the size of the training set is already large enough that the GAN is able to extract valuable information. For an image with a size of $650 \times 650$ pixel, that corresponds to about 4 $km^2$ and a patch size of $321 \times 321$ pixels, the training roughly contains $108k$ individual

training samples.

### Road Network Generation

After the training of the GAN, the discriminator component $D$ can be omitted as it is not necessary anymore. For the synthesis of novel road networks, only the generator $G$ and its learned parameters $\theta^{(G)}$ are necessary. Synthesizing a novel image that contains a road network structure is fairly easy. We use the generator component $G$ and feed it with a volume $z$ where $z \in \mathbb{R}^{m \times m \times d}$ where $m$ represents the spatial resolution and $d$ the dimension of the randomly sampled vectors that will be inserted at each spatial location. Each $d$-dimensional vector is drawn from a simple distribution. As it was already described in Section 4.2.3 and 4.2.4 the generator maps its input to an image $x' \in \mathbb{R}^{w \times h \times 1}$, where $w$ and $h$ represent the width and height of the image, respectively. For all the examples shown in the current chapter, we sampled values from a uniform distribution and limited the values to stay within $[-1.0, 1.0]$.
One advantage of the *SGAN* technique is that the generator network contains only learned filters and has no fully connected layer, that would fix the network topology. This allows that the spatial resolution of $z$ can be varied after the training. Thus, a multitude of different resolutions can be achieved by changing the spatial resolution of $z \in \mathbb{R}^{m \times m \times d}$. The only parameter that is fixed is the depth $d$.

### Road Network Post-Processing

The images produced by the generator component $G$ contain a road network encoded as pixel intensities. In order to use the resulting road network in GIS applications or in a road network analysis task, we need to transform the gray-scale image intensities back into a road network graph. For this purpose, we apply a post-processing to the synthesized images.
**Image post-processing:** The gray-scale images produced by the generator network contain pixel intensities in the range $[0, 255]$. In a first step, we threshold the gray values at 127 in order to produce a binary image where pixels set to *true* represent the presence and pixels set *false* represent the absence of a road. Applying the threshold might produce undesirable isolated pixels and also small cusps along road regions. In order to get rid of these artifacts, we apply a morphological erosion operation. However, the produced result might still contain small holes or road regions that do not touch within a local radius of up to five pixels. In order to close such small gaps, we apply five steps of morphological dilation. For all morphological operations, we use a $3 \times 3$ structuring element with the shape of a cross. The obtained initial road network, however, contains road regions that are too thick to extract an initial road graph. Therefore, we thin out the result and

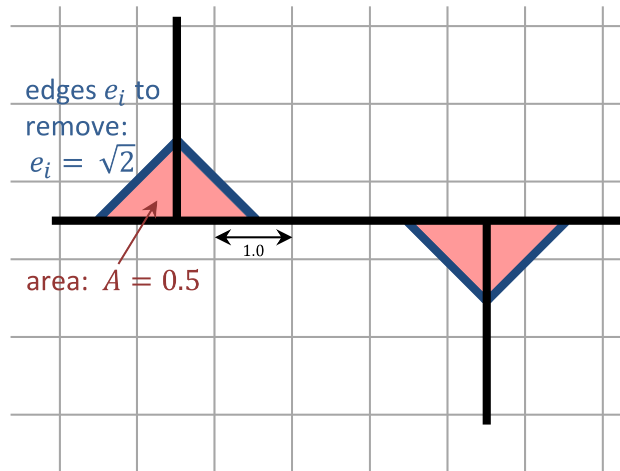



Figure 4.14: Illustration of the block artifacts that result from the graph construction.

extract a skeleton from the cleaned binary image using the algorithm from Zhang et al. [ZS84].

**Road Graph Construction:** We utilize the pixel-skeleton from the previous step to construct an initial graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ representation of the synthesized road network, where $\mathscr{V}$ are its vertices and $\mathscr{E}$ are its edges. In order to construct $\mathscr{G}$, we add a node $V_i$ to $\mathscr{V}$ for each of the skeleton pixels. Next, we examine the 8-neighborhood of each $V_i$ in the image. For each skeleton pixel $V_j$ inside the 8-neighborhood of $V_i$, we add an edge $E_{ij} = (V_i, V_j)$.

### Road Network Post-Processing

**City Block Cleanup:** The graph construction from the pixel skeleton produces regions within the road network that have a very small area of 0.5 square pixels (see Figure 4.14), which are removed in a first step. The regions within a road network graph are typically called city blocks. Strictly speaking, a city block is a region within the graph $\mathscr{G}$, that is enclosed by a set of road segments and might contain dead-end street segments. In order to identify these small regions, we first compute all the city blocks of the graph. As the graph is a directed graph and embedded in $\mathbb{R}^2$, the city blocks can be computed by determining the minimal cycles of the graph by computing loops of edges. Next, we filter out blocks with an area of 0.5 pixels. These artifact blocks can be removed by identifying and removing their longest edge, which has a length of $\sqrt{2}$.

**Road courses smoothing:** Another artifact produced by constructing $\mathscr{G}$ from the image raster are jagged edges. In order to smooth these in the final graph, we

extract a set of street chains $S = \{S_i\}$ from the graph. Each $S_i = \{V_0, \ldots, V_n\}$ consists of $n$ nodes, while the degrees of $V_0$ and $V_n$ are constrained by $deg(V_0) \neq 2$ and $deg(V_n) \neq 2$. From each of the $S_i$, a polyline $P_i = \{p_0, \ldots, p_n\}$ with $n$ positions is built. A smoothed version of the positions can be obtained by applying 5 steps of local averaging $\hat{p}_i = \frac{1}{4}p_{i-1} + \frac{1}{2}p_i + \frac{1}{4}p_{i+1}$ to the $p_i$'s with $i \in [1, n-1]$, and replacing the original $p_i$'s with their smoothed version $\hat{p}_i$. Finally, we additionally straighten the road courses, by removing superfluous nodes using the Douglas Peucker simplification [DP73]. We allow to remove nodes that deviate up to 3 meters from a straight line. As the generator $G$ produces many short dead-end streets, we remove small dead-end street chains with a total length of up to 25 meters.

### 4.2.5 Case Study: Road Network Types

In order to showcase the versatility of our road network synthesis approach, we evaluate our approach on a set of challenging test cases. We composed a collection of real-world road network as well as synthetic road network examples (see Figure 4.15 a)-d)). The real-world examples were taken from OSM, while the synthetic ones were taken from [MIS17]. For all the examples shown in here, we used a patch size of $n \times n = 321 \times 321$ (cf. Figure 4.20) pixel during the training procedure. That size captured the local structures present within the different road networks used for the evaluation. Furthermore, we used only a single road network image from which patches were extracted. We synthesized two examples for each road network shown in Figure 4.15 a)-d). In our evaluation, we investigate the visual appearance of the generated results and analyze the similarity in terms of road networks measures such as area, compactness and aspect ratio of the city blocks by comparing the resulting distributions.

### Visual Evaluation

**Irregular: Synthetic** As a first test case, we considered a synthetic road network (see Figure 4.15a). The major characteristics of this road network are blocks of different sizes with and without dead-ends, and blocks of similar size, that form small groups. Nearly all blocks have a rectangular shape up to a few exceptions. Figure 4.16a and 4.16b show road networks generated by GAN model after passing our post-processing pipeline. It can be noticed, that the generated results contain blocks similar in shape and size when compared to the original network. Notice, that the results even contain the small groups of nearly square shaped blocks that are present in the original network. Larger road courses are present in the examples, although they have different curvatures when compared to the original network.
**Irregular: San Marco** Next, we evaluated a street network patch from a village in Italy (see Figure 4.15b). A major characteristic of that network is its large amount of small city blocks in comparison to only a few larger ones. Generated samples of this network type are depicted in Figure 4.17a and 4.17b. Both samples contain a significant number of small blocks when compared to the number of medium-sized and large city blocks. It is also noticeable that smaller blocks are located next to each other. Furthermore, the result contains large-scale structures such as connected road courses that separate groups of smaller blocks. Another produced sample visualized using CityEngine[Esr17] can be seen in Figure 4.26.
**Irregular: Berlin** In contrast to the previous example, the network that is shown in Figure 4.15c is composed of a significant amount of larger, mainly square or rectangular shaped blocks. Only a few blocks are irregularly shaped and even contain dead-ends. The generated samples shown in Figure 4.18a and 4.18b contain a significant amount of nearly square shaped blocks and rectangular shaped blocks. It can be recognized that the generated networks also contain irregularly shaped blocks and even L-shaped blocks not being present within the example network.
**Suburban: Synthetic** Next, we show results generated from a synthetic network of a suburban region with structures mainly found in rural regions of the US (see Figure 4.15d). A major property of such network types is the presence of curved road courses. Our produced results shown in Figure 4.19a and 4.19b contain these typical curved roads shapes.

### Statistical Evaluation

Apart from the visual comparison of the results, we performed an evaluation of graph measures computed on the synthesized road networks and the original road networks. These measures include the *cityblock area*, the *compactness*, i.e., the ratio between block area and its minimum bounding box, and the *city block as-*

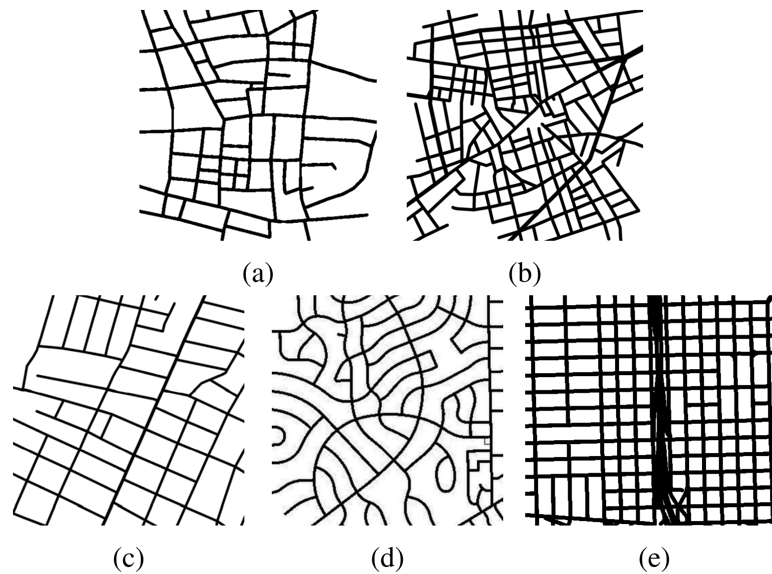



Figure 4.15: Overview of the different road network styles used in our case study: (a) Synthetic irregular, (b) Cellino San Marco irregular, (c) Berlin irregular, (d) Synthetic suburban, (e) Portland with highway ramps.

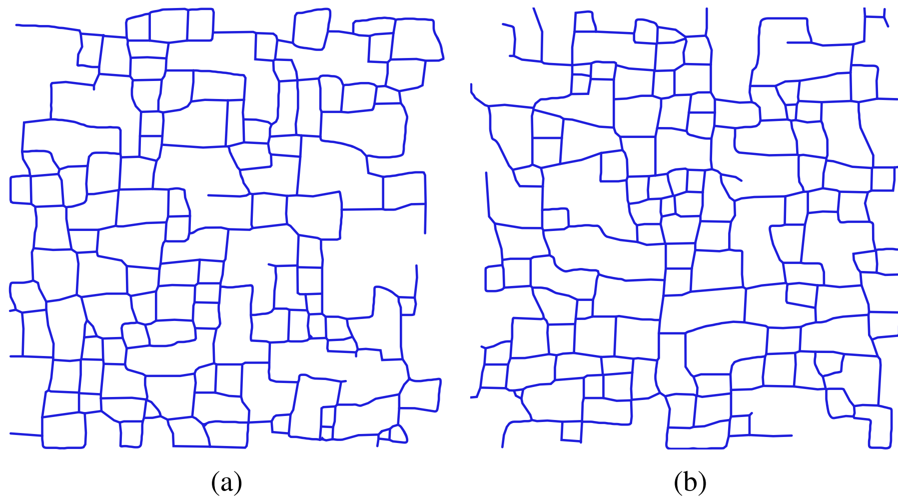
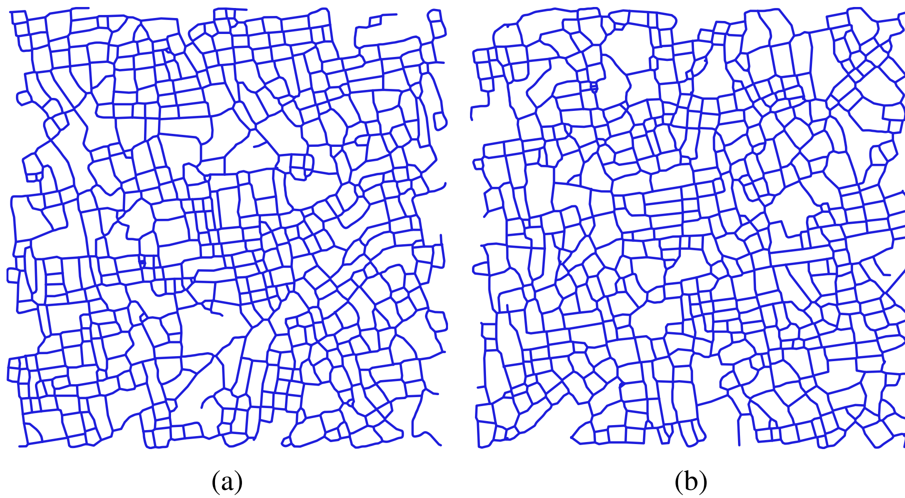


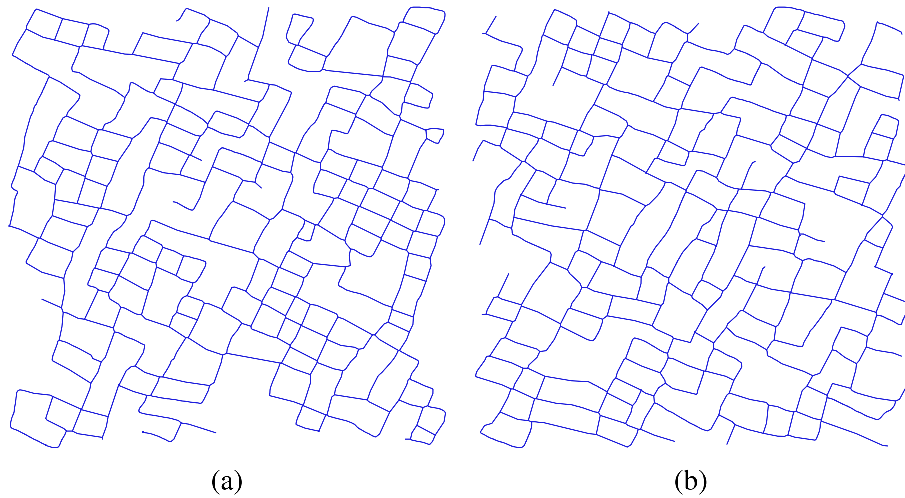

Figure 4.16: Two road networks generated using the trained GAN model after passing the postprocessing pipeline. Both networks contain blocks that are similar in shape an size when compared to the road network used for training (see Figure 4.15a).

*pect ratio*, i.e., the ratio between the shorter and the longer side of the minimum bounding box.



(a) (b)

Figure 4.17: Two road networks generated using the trained GAN model based on the road network shown in Figure 4.15b. As it can be noticed the network consists of different sized blocks that even arrange in groups of blocks of similar size.



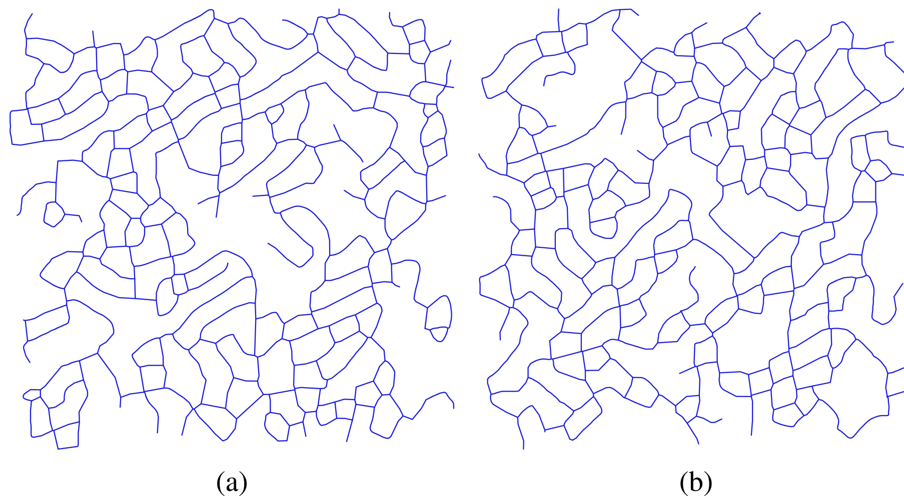
(a) (b)

Figure 4.18: Here we show two road network samples that were generated using the GAN that was trained on image patches from the road network illustrated in Figure 4.15c. Both results consist of a large amount of square and near rectangular blocks when compared to the original road network. Even irregular shapes such l-shaped city blocks occur in both result road networks.

**Irregular: Synthetic** Figure 4.21 compares the graph measures between the synthetic irregular network shown in Figure 4.15a with the ones obtained from our synthesized results. While the distributions of the block area and the compactness have a similar shape, the aspect ratio distribution varies as the generated result contains much more variation of rectangular shaped blocks than the original road network.

**Irregular: San Marco** In this result (see Figure 4.15b), the distributions of block area, aspect ratio and compactness are similar to each other(see Figure 4.22). The resulting network mostly consists of small city blocks as illustrated by the block area distribution. Both the original and the generated road network contain a significant amount of nearly rectangular blocks (see compactness). As the aspect ratios within the generated network are also similar, thus, the learned model has captured the characteristics of the original network.

**Irregular: Berlin** In Figure 4.23, we illustrate the distributions for the Berlin example shown in Figure 4.15c. While the block area and the aspect ratio of the blocks found in the generated example tend to be similar, the compactness varies more than in the previous examples. As the streets in the produced network are not perfectly straight anymore, the compactness of the blocks deviates from being close to 1.0.



(a) (b)

Figure 4.19: Curved road structure as common in suburban regions of US-cities can be faithfully reproduced using the GAN model trained from image patches of the original network depicted in Figure 4.15d. Large-scale features, i.e., large curved road structures, can only be reproduced up to the size context used for training the network.

**Suburban: Synthetic** For suburban networks such as the one shown in Figure 4.15d, the distributions of block area and aspect ratio differ, while especially the aspect ratios within the generated network have a few spikes (cf. Figure 4.24). However, at a larger scale, the overall shape of the distribution is similar. As this road network type contains large-scale structures such as curved roads that pass through the whole network, the chosen context size cannot capture these. Thus, the generated network will suffer from these missing global properties. This leads to a structurally different generated road network which is reflected by the distributions of the different graph measures.

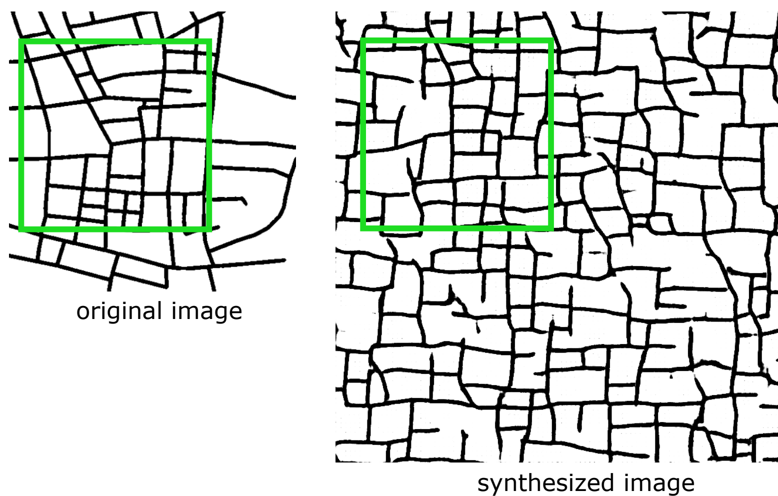



Figure 4.20: Illustration of the context size using during the training stage. Left: Original image with the overlaid extent of the training image. Right: Generated sample with an overlay of the training image size.

### Limitations

**Large-scale structures and ramps**. We noticed that our approach cannot successfully handle road network patches that contain highway ramps and networks that contain street lanes that are located very close to each other, as illustrated in the road network example taken from Portland (see Figure 4.15e). When the road network is rasterized nearby lanes will be merged and form even thicker lanes. If highway ramps are present, additional pixel blobs are introduced as illustrated in the synthesized example shown in Figure 4.25. It can be noticed that the grid-like road pattern is faithfully reproduced. However, due to the thick lanes and the limited context size (see Figure 4.20) the highway structures present in the training data cannot be recovered successfully. Instead, thick road structures occur on the left border (cf. green arrows), and blob-shaped artifacts are scattered

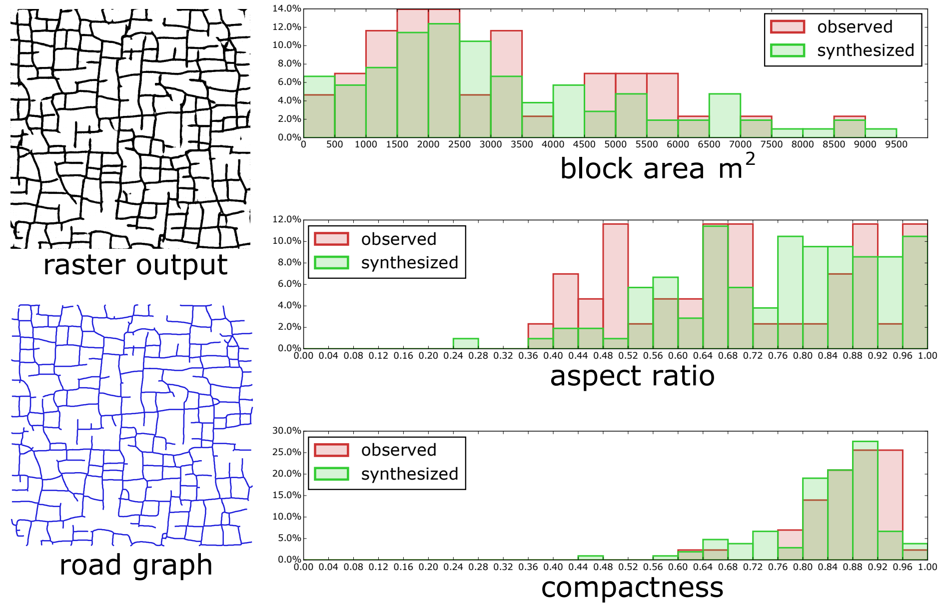



Figure 4.21: Statistical evaluation of Synthetic irregular

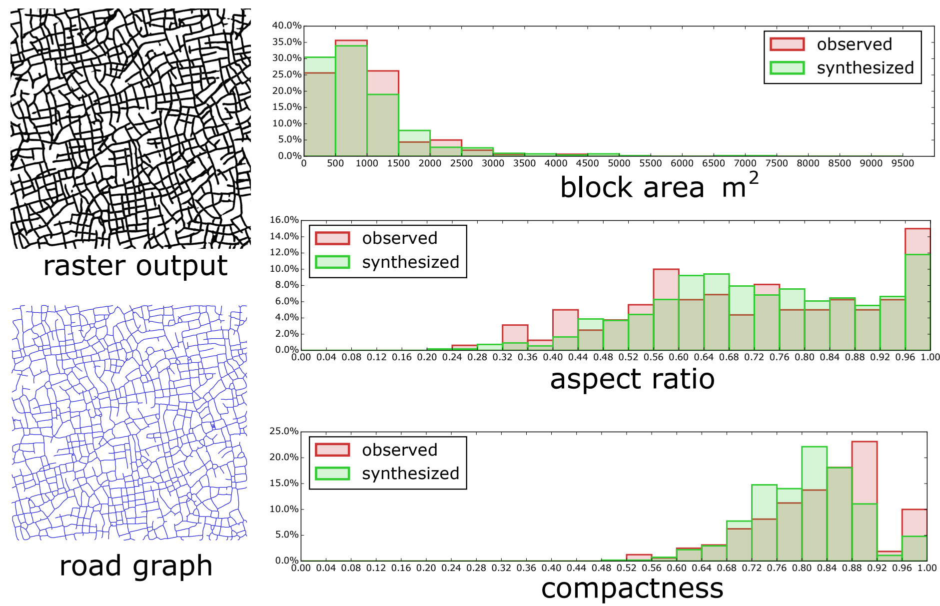



Figure 4.22: Statistical evaluation of Cellino San Marco

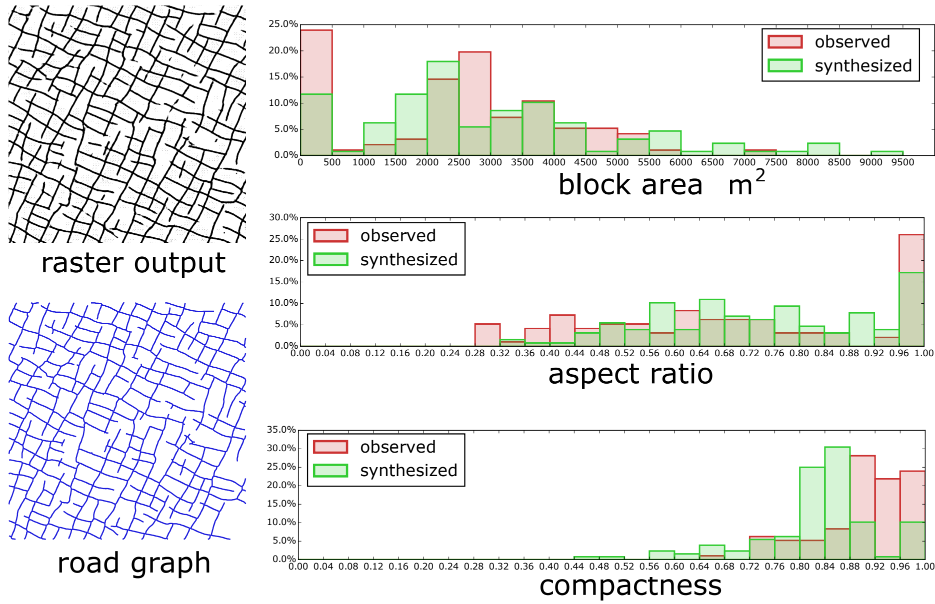



Figure 4.23: Statistical evaluation of Berlin irregular

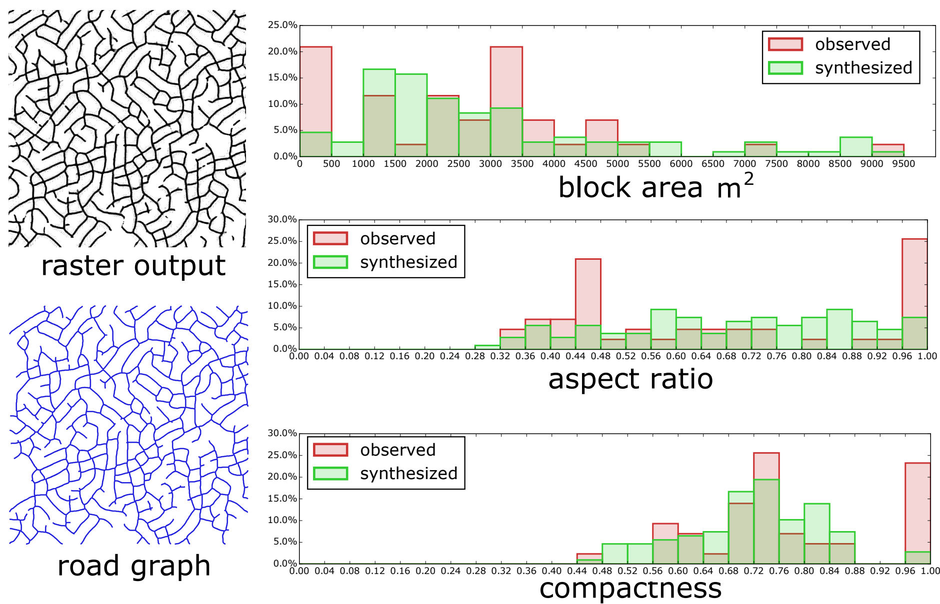



Figure 4.24: Statistical evaluation of synthetic suburban

over the synthesized example (cf. region surrounded by green ellipses). When the post-processing is applied, these artifacts will be alleviated; however, irregularly shaped blocks will be present in the final road network.

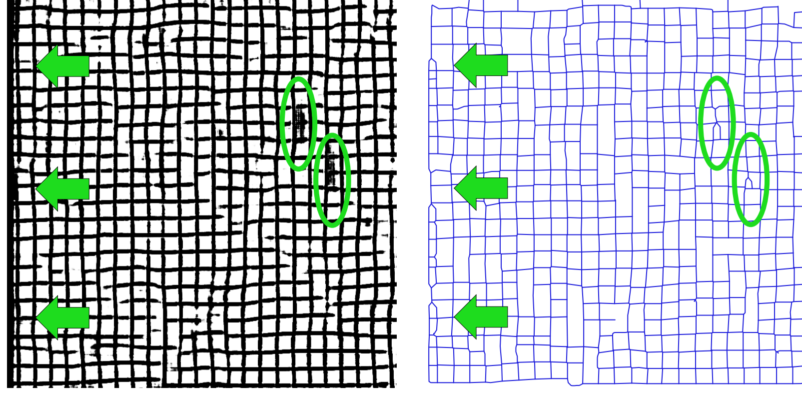


Figure 4.25: In case of nearby located highway lanes and highway ramps, the GAN fails to capture these properties. This leads to blob-like artifacts in the generated samples.

**Deadend roads**. All the synthesized examples contain much more dead-ends when compared with the number of dead-end streets present in their corresponding original road network. This might be due to the patch-based training procedure. Each patch that is used for training contains virtual dead-end street segments that abruptly end at the patch boundary.

### 4.2.6 Implementation Details

The algorithms are implemented in Python, and we used GAN implementation of [JBU16] as a basis for learning the different road network models. However, the original implementation was adapted in order to consistently support single-channel images. The GAN model for the different road networks is trained on a single NVidia TitanX (Pascal). Each epoch takes 100 iterations with a batch size of 64 and takes about 90 seconds to compute. We trained all the models for at least 100 epochs and decided from a visual examination of samples taken from various epochs which model to choose. The overall training is done in an offline step that takes up to 3 hours. The single steps of the online synthesis step take up to a few seconds. In more detail, the generation of a sample of size $769 \times 769$ pixels produced from a tensor $z \in \mathbb{R}^{25\times 25\times 100}$ sampled from $p_z(z)$, takes on average 0.08 seconds on a single NVidia TitanX (Pascal). The postprocessing steps are performed on an Intel Core-i7 5820K, with only a single core in use. Each step takes: for *graph construction*: 1.5$s$, for *block computation*: 1.6$s$, for *simplification*: 2.0$s$ and for *deadend removal*: 0.02$s$ in average.

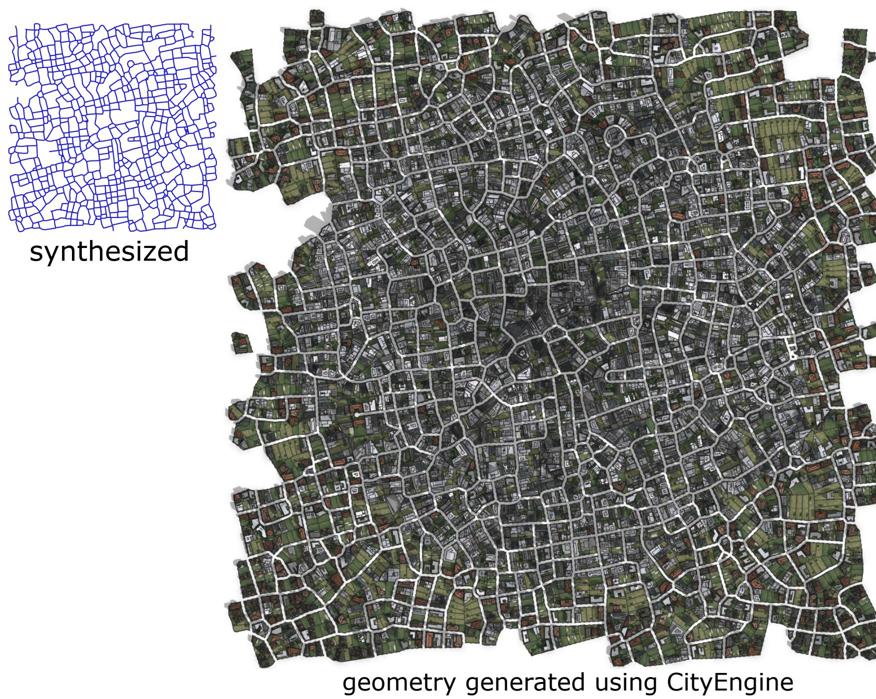
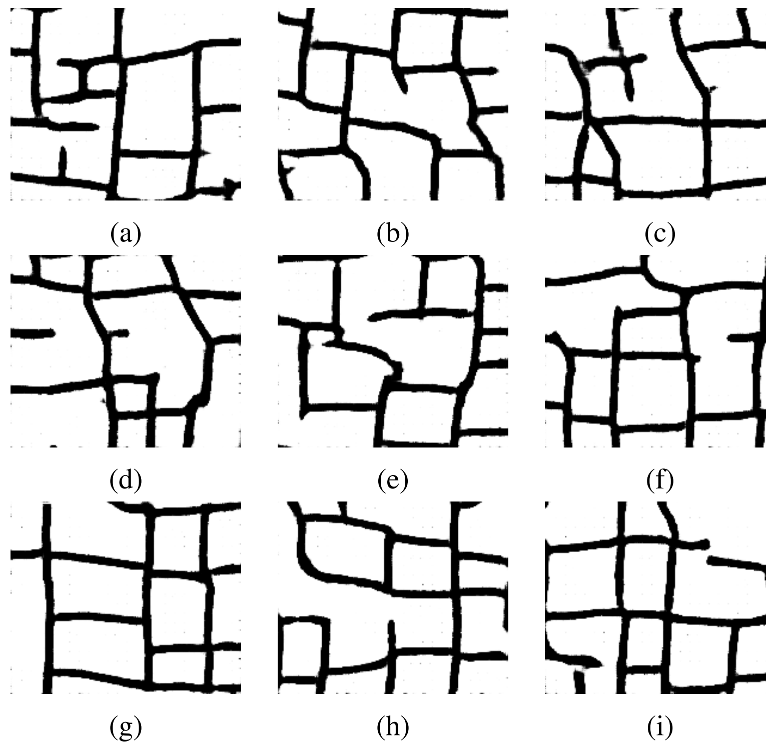



Figure 4.26: The resulting road network is directly usable in urban planning tools such as CityEngine.

### 4.2.7 Analysis and Comparison

There exist a variety of road network synthesis algorithms, that already produce realistic road networks and road structures including the one presented in chapter 4.1. The proposed approach mainly differs by replacing the typically custom-tailored algorithm with a generative model. The task of encapsulating expert knowledge within complex algorithms is transferred to the computer, especially the neural network that is trained to capture the statistics and structure present in the exemplars. We use images to feed information into the neural network. Thus we lose important information about the street topology and structure, that is typically exploited by existing techniques [NGDA16a, YWVW13]. However, the generated samples show similarity to the exemplars at smaller scales such as city block groups. Large-scale structures cannot be faithfully reproduced, because, the neural network only 'sees' small patches that were cut out at random positions. The integration of different street levels might be a good choice to capture road courses at larger scales. However, we leave this for future work.

A significant advantage of the GAN architecture used is the absence of any fully-connected layout within the neural network. Thus it is not restricted to producing outputs of a fixed size. By varying the spatial resolution of $z$ it is possible to



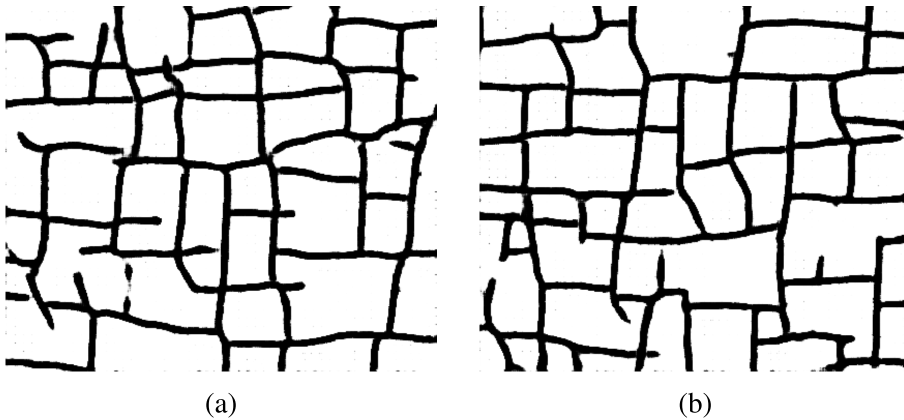
(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 4.27: Synthesized road network images patches: These patches use the generator trained on the example road network shown in Figure 4.15a. Each patch was synthesized using a different sample $z_i$ with a spatial resolution of $6x7$ resulting in images with a resolution of $193x161$ pixels. Note small-scale structures such as city blocks are present in generated patches.

synthesize small patches (see Figure 4.27) as well as large patches (Figure 4.28). These patches might serve as open templates similar to Nishida et al. [NGDA16a] or these patches could also serve as 'fill' street patches and might help to synthesize the street layout at higher hierarchy levels in our approach that was presented in chapter 4.1.

### 4.2.8 Limitations

During the evaluation of our pipeline, we identified several limitations. First, structures like roundabouts, highway ramps and also roads that are very close to each other are not sufficiently captured during the training. This means that roundabouts or highway ramps cannot be successfully synthesized with our approach. Second, currently, we consider all *highway* categories as part of the same street



(a) (b)

Figure 4.28: Generated road network patches. These patches use the generator trained on the example road network shown in Figure 4.15a. In contrast to Figure 4.27 a spatial resolution of $12x14$ (image size: $417x353$ ) was used for each sampled $z_i$. Note that the structure found in the example network are also present within these patches.

level. We did not succeed in learning models for different street levels. Thus, we decided to perform the experiments using only a single street level (cf. Section 4.2.4). Typically, a road network naturally splits into multiple street levels such as *major* and *minor* roads. Thus, it is necessary to perform an in-depth evaluation of multiple street levels in future work.
Furthermore, large-scale road courses are typically present in every road network. Although, these structures are rudimentary present in the synthesized examples shown in Section 4.2.5, our post-processing step lacks an additional step to enforce such large-scale structures consistently. One possibility to address this issue, would be fitting curves to individual road courses and enforce global constraints such as parallelism.
Another limitation is that we currently have no control over the output of the generator. In real road networks, the road courses are specifically planned to fulfill specific requirements regarding land use or terrain. Furthermore, the urban planner might also incorporate existing objects into its road design decisions. As our approach is a very first step towards using GANs for road network generation, we did not incorporate such external constraints. However, such constraints are necessary to steer the output of the generator $G$, and we leave this for future work.

# CHAPTER 5

# Example-Based Cityblock Layout Synthesis

## 5.1 Example-Based Cityblock Layout Transfer

### 5.1.1 Motivation

The techniques for synthesizing individual road networks that are discussed in section 4.1 and 4.2 have shown that the example-based methodology is a powerful technique to transfer the style of real-world examples to virtual urban environments. So far, we have only considered the generation of road networks. In order to produce a more detailed and realistic city layout, the empty regions within the road network - the city blocks - need to be filled with building footprints and 3D building models. In here, a building footprint represents the outline of an individual building at ground level. This section focuses on synthesizing city block layouts, i.e. we fill the empty regions present within the street networks with realistic building footprint layouts. The major goal is following the example-based modeling metaphor and re-using city block layouts from real-world cities and use them to transfer their individual building footprint layout to a virtual urban environment. We believe that this technique is particularly well-suited for the envisioned task as (1) we can find a huge amount of real-world city block layouts in GIS repositories on the Internet, (2) these city block layouts mirror the style of the building arrangements found within the real world, and (3) optionally provides rich annotations about building usage or additional details such as positions of phone and letterboxes, trees, or even recycling containers and playgrounds. Online GIS mapping services such as *OpenStreetMap*[Ope17a] seem to be a suitable choice as a data source because they provide extremely detailed city layouts from

all over the world that are covered with detailed footprint layouts (see Figure 5.1). In the following, we will define the problem setting and provide detailed information about the technique we propose to fill real-world city block layouts into city blocks of virtual road networks.

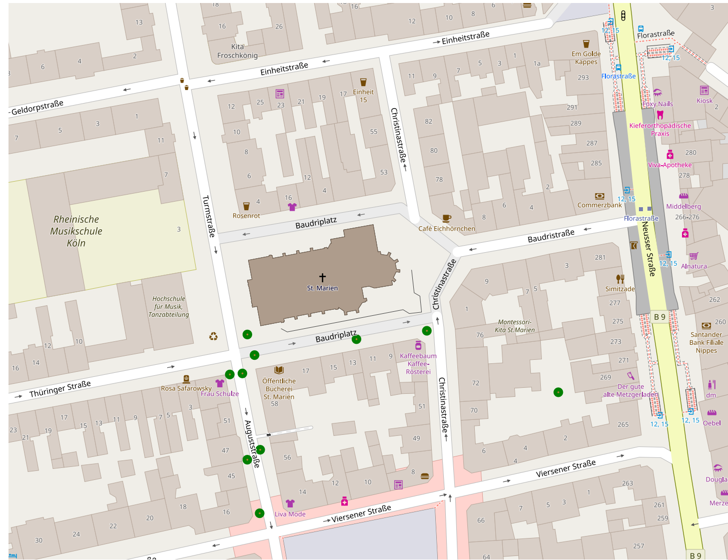



Figure 5.1: Layouts of city blocks as they can be found at GIS repositories such as OpenStreetMap. Here a part of Cologne is depicted, that contains detailed building footprints, tree locations (green circles) and other rich annotations such a building type, i.e., coffee shop, or pharmacy.

### 5.1.2 Problem Definition and Overview

In this chapter we investigate the re-use of real-world city block layouts to enrich the details of virtual city layouts, i.e., we use a database of city block examples and use them to infill the empty regions present within a street network, that are completely enclosed by street segments. Thus, we pursue the direction that has been used for the hierarchical generation of road networks discussed in section 4.1. In contrast to existing city block layout algorithms that subdivide the empty city blocks into a set of smaller entities called parcels (see section 2.2.1 for a detailed review), we focus on the transfer of the building footprints from a real-world city block into a similarly sized and shaped virtual city block by copying the original footprint shapes into the empty city block. One significant advantage of such a strategy is that we can resort to the huge variety of individual footprint layouts that can be found in the real world. In addition, to the variation present within the building footprint layout of the city block itself, the individual footprints contain

large variations in their shape which in return increases the realism of the overall layout.
Next, we briefly discuss the overview of the example-based footprint synthesis algorithm. The input to the city block layout algorithm is a street network that might come from different sources such as the road network generators from Section 4.1 and 4.2, from OpenStreetMap[Ope17a], or a procedural road network generation system such as CityEngine[Esr17].

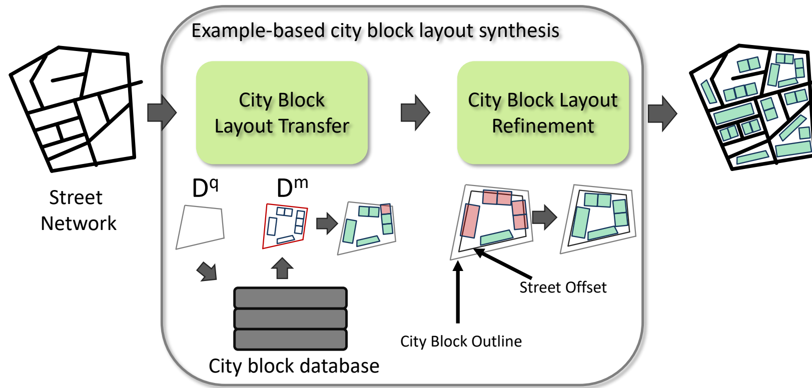



Figure 5.2: Our footprint and 3D building synthesis pipeline

When we speak of a city block, we speak of a domain $D \in \mathbb{R}^2$ where $D = (B, \mathscr{P})$ is a tuple composed of a boundary $B$ and a set of building footprints $\mathscr{P} = \{P_1, \ldots, P_n\}$. From a given street graph $G$, a set of city blocks $\mathscr{D}^q = \{D_1^q, \ldots, D_n^q\}$ is extracted that will be filled with building footprint arrangements layouts taken from real-world city blocks $D^m$ that are typically stored in a database. We use superscript $q$ in order to indicate that the city block is used as query object to retrieve a suitable real-world city block, and use superscript $m$ to indicate that this block was retrieved as a suitable match from the city block database. The set $\mathscr{D}^q$ is extracted from the provided street graph $G$ by extracting minimal cycles. These can be extracted by computing closed walks of street segments, i.e., edges of $G$. The resulting set of cycles represent the outline $B$ of the individual city blocks $D_i^q$ used as query. Each city block $D_i^q$ within the virtual street network is used to retrieving a real-world city block $D^m$ from a database of example city blocks, that is similar in shape and size. We use a database of example city blocks originally taken from different real-world cities containing a huge variety of different sizes and shapes. After the retrieval step both the query object $D^q$ and the best matching candidate $D^m$ are aligned and the footprints located inside $D^m$ can be transferred by copying their shapes into the city block $D^q$ originally used as query object. After transferring the retrieved layout to $D^q$, individual $P_i$'s might be partially located outside the outline $B$ of $D^q$ or are partially located on the street. These building foot-

prints are marked as 'conflicted' and are resolved by employing a physics-based simulation. The individual steps of the algorithm are depicted in Figure 5.2.

### 5.1.3 City Block Layout Transfer

As mentioned above, we want to pursue the direction that has been used for the hierarchical generation of road networks (see Section 4.1), where we used shape matching in order to identify viable candidate fragments from the set of exemplars. In order to compare the shape of a $D^q$ against the city block shapes that are stored within the example database, we use the *Shape Context* introduced by Belongie et al. [BMM00]. However, in contrast to section 4.1 we drop the term that took the relief into account for the evaluation of the technique and the results shown in this section because we did not have access to detailed terrain information for all of the used cities especially the German ones. However, it can be incorporated as it was done in chapter 4.1 when access to high-resolution terrain data is available.. Instead, we focus only on the comparison of the individual shapes of the city blocks. In order to use the Shape Context for comparing city blocks, we uniformly sample point sets along the boundary $B$ of a query city block $D^q$ and the boundary $B$ of the city block $D^m$ from the example database resulting in two point sets $P = (p_1, .., p_n)$ for $D^q$ and $Q = (q_1, ..., q_n)$ for $D^m$. As the *Shape Context* characterizes the local neighborhood of a sample point $p_l$ in terms of a histogram over the distances from $p_i$ to neighboring outline points it allows for efficient comparison between two shapes. The dissimilarity of two Shape Contexts of points $p_l$ and $p_k$ is denoted as $C^S_{lk}$ and the overall distance between two shapes $P = (p_1, .., p_n)$ and $Q = (q_1, ..., q_n)$ according to the sets of underlying Shape Contexts is determined by the computation of a (cyclic) permutation $\Pi(Q) = (q_{\pi(1)}, ..., q_{\pi(n)})$, such that the resulting matching costs $C(P,Q) = \sum_{i=1}^{n} C^S_{i,\sigma(i)}$ are minimized. As discussed in Section 4.1 the similarity between two city blocks is computed, by minimizing the matching cost between all pair permutations $C_{ij}$, which boils down to computing the least cost path through the cost matrix $C$. This can be efficiently achieved by the *Dynamic Time Warping* (DTW) algorithm [SC78]. The computation of Dynamic Time Warping between a query city block $D^q$, and all the $D^m_j$ present in the example database would be too costly. Therefore, we first prune inappropriate city blocks that are too different in area, outline length and aspect ratio. For determining the best suitable candidate by using the first $c = 10$ candidates from the pruning step and compute the similarity measure between $D^q_i$ and the $D^m_j$'s using *Dynamic Time Warping*.

Having identified a suitable city block candidate $D^m$, that is the most similar according to the $c$ candidates, we can now transfer the building footprints $\mathscr{P}$ from the city block $D^m$ to the city block $D^q$. Typically, both the query city block $D^q$

and the retrieved candidate city block $D^m$ are not aligned and live in different coordinates systems. However, we can use the corresponding points to align them. Luckily, we already computed corresponding points $P = (p_1, .., p_n)$ for $D^q$ and $Q = (q_1, ..., q_n)$ for $D^m$ during the computation of the least cost path using the Dynamic Time Warping algorithm. We use these corresponding points to compute an alignment $(R, t)$, where $R$ is a rotation, and $t$ is a translation, that aligns $P$ and $Q$ in least squares sense. Such an alignment can be computed by solving the following optimization problem

$$(R, t) \underset{R \in SO2, t \in \mathbb{R}^2}{\operatorname{argmin}} \sum_{i=1}^{n} \|(Rq_i + t) - p_i\|^2$$

.

After computing the alignment, we can now transfer the footprints $\mathscr{P}$ located in the city block $D^m$ into the city block $D^q$, by employing the computed rigid transformation to each individual footprint. The matrix $R$ contains the rotational part of the transformation and $t$ is a translation. Applying it to all footprints $\mathscr{P}$ will not lead to overlaps between individual footprint, but might lead to two types of problems, (1) the transferred footprint $P_i$ is located outside or partially outside the outline $B$ of $D^q$ and (2) the footprint $P_i$ is located on the street area of $D^q$. Both cases might occur when the shape difference between $D^q$ and $D^m$ is large. We handle both cases in order to ensure that the transferred footprints $\mathscr{P}$ form a valid layout, which means that no footprint is partially located either outside the outline of $D^q$ and is partially located on the street area. In the case of (1), we simply remove the affected footprint from the set $\mathscr{P}$ in $D^q$. In the case of (2), we mark the footprint as conflicted and perform a layout refinement step using a physics-based simulation in order to resolve such conflicts (see section 5.1.4).

### 5.1.4 City Block Layout Refinement

The transfer of the building footprints into the query city block $D^q$ may lead to undesired side effects as described in Section 5.1.5. These effects include building footprints that overlap with nearby located street segments. In order to reduce these side effects and resolve such conflicts, we employ a constraint optimization, inspired by physics simulation. Essentially, the footprints are modeled as rigid bodies, and collision between street segments and them are resolved using position based dynamics [MHHR07]. To accelerate overlapping tests, each footprint is approximated using a *Bounding Volume Hierarchy* (BVH). Before the optimization starts, conflicted footprints (see red colored areas in Figure 5.3) and non-conflicted footprints (see dark green colored areas in Figure 5.3) inside $D^q$ are detected in a first step. All footprints that do not partially overlap the street area are marked as non-conflicted and stay fixed at their current position.

Figure 5.3: Conflicted building footprints (grey) that are partially placed on road and pavement

Figure 5.4: Partially overlapping conflicts are resolved utilizing our constraint optimization scheme

The footprints are represented as an undirected graph structure $P = (C, W)$ where $C$ represents the corner points of the footprints and $W$ denotes the edges connecting the footprint vertices typically representing walls. Each edge $w_i$ stores its desired target edge length $l_w^t$.

If a footprint vertex $p_i$ is located outside the offset polygon of the street or inside another footprint, a collision impulse is simulated by setting the building point location $p_i'$ to the outline of the offset polygon or the footprint. In a Gauss-Seidel relaxation scheme, using explicit *Verlet Integration* the position of the new building points is iteratively changed such that none of the soft constraints is violated, while the difference $dl_w = \|l_w^t - l_w^c\|^2$ between target edge length $t_w^t$ and current edge length $l_w^c$ is minimized.

### 5.1.5 Example-Based 3D Building Placement

After the synthesis of the building footprint layout into the city blocks of a provided street graph, we now describe how we add details such as 3D building models to a synthesized city layout in order to increase its details and its realism further. In contrast to chapter 6, where we focused on the synthesis of individual buildings we re-use already modeled 3D buildings, that might be placed as whole on top of similarly sized footprints. The main goal here is first trying to re-use the buildings by comparing building footprints and synthesized footprints. The process is typically much faster then synthesizing individual buildings. In addition, we need segmentations of the buildings into basic building blocks including semantic annotations in order to apply the technique proposed in chapter 6. Such data is difficult to create and might not be available for existing large building databases. Therefore, we use a large repository, Trimble 3D Warehouse [Tri17], that contains a huge amount of detailed building collections that are publicly available. This repository contains 3D buildings models from all over the world containing a large variety of geometric details and buildings shapes. In order pursue the example-based modeling metaphor from section 5.1.3 and place 3D building models on top of the synthesized building footprints by comparison of the footprint shapes, it is crucial to have access to a building footprint for these 3D models, that represent their individual shapes at the ground level. However, the 3D buildings from such an online repository typical consist of 3D geometry and optional textures and do not carry any building footprint. In order to successfully synthesize 3D buildings into a city block layout, two essential steps are necessary: (1) the extraction of the building footprint for the individual 3D buildings, and (2) the placement of the 3D buildings on top of synthesized footprints.

#### Extraction of Footprints of 3D Buildings

Next, we discuss how we extract the 2D outline of the 3D model in order to allow the envisioned application. Instead of using a geometric approach, we employ a robust image-based technique to compute the footprint of the 3D building using rasterization. We render the building from the top view into a binary image. In such an image, the pixel intensities encode the presence or absence of the 3D building. The extraction of contours from binary images is well understood; therefore, we rely on the technique of Suzuki [S+85] to extract the contours. We make the assumption that a building typically consists of a single contour and thus the largest connected component of the extracted contour set, represents the 2D outline of the building footprint. The extracted outline typically contains a large number of redundant points along the walls. Therefore, we further simplify the footprint polygon employing a feature preserving chain approximation algorithm

[TC89]. The resulting outline $P$ consists of a set of $n$ points $P = \{P_1, \ldots, P_n\}$ and is stored along with its corresponding the 3D model $M$ in a *3D Buildings* database as a tuple $H = (M, P)$. Such a tuple $H$ that represents a 3D model $M$ and its corresponding building footprint $P$, illustrated in Figure 5.5). Please note that this step needs only performed once for each building in a pre-processing step.

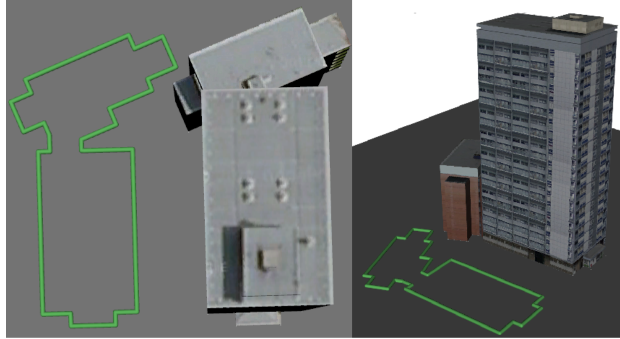


Figure 5.5: Extracted footprint (green) from the illustrated 3D building. The footprint is extracted by rendering the building from its top view into a binary image. Then we extract the boundary and simplify the largest connected component using a feature preserving chain approximation algorithm. The extracted footprint is finally stored along with the 3D building model in the 3D building database.

#### Placement of 3D Building Models

The placement of buildings is performed after the *City Block Layout Refinement* step from Section 5.1.4. The overall goal is to place a 3D model on top of each of the synthesized footprints $P$ present within the set $\mathscr{P}$ of each city block $D_i^q$ of the set $\mathscr{D}^q = \{D_1^q, \ldots, D_n^q\}$. Next, we discuss how we perform the placement of an individual building. In order to place 3D buildings on each footprint present within the city layout, this procedure needs to be repeated multiple times and can be performed in parallel. In order to place 3D buildings on top of the synthesized building footprints, we perform exactly the same steps, that were performed for the retrieval of viable city blocks $D^m$ in section 5.1.3. Instead of pruning, retrieving and aligning the outline $B$ of the city blocks $D^q$ and $D^m$ we now use the footprints $\mathscr{P}$ as query objects $P^q$ in order retrieve suitable set of building candidates $\mathscr{H}^m$ containing $c$ building tuples $\mathscr{H}^m = \{H_1^m, \ldots, H_c^m\}$, that have similar area, outline length and aspect ratio. The shape comparison using *Shape Contexts* is now performed between the 2D outline of the query footprint $P^q$ and the footprints $P_i^m$ of the candidate buildings $H_c^m$. The retrieval of the buildings can optionally be constrained by an architectural style map that is provided as an image

where the pixel intensities encode individual buildings styles such as residential buildings, skyscrapers, or commercial buildings.
The diversity of the 3D buildings and thus the shape of the buildings footprints is typically much smaller than the shape of the building footprints within the synthesized layout. In order to constrain the placement of buildings where the shape difference between $P^q$ and $P_i^m$ is large, we perform an additional check before we allow the 3D building to be placed into the synthesized layout. First we compute the intersection

$$I_{P^qP^m} = P^q \cap P^m$$

between $P^q$ and $P_i^m$.
Let

$$O^q = \frac{A(I_{P^qP^m})}{A(P^q)}$$

be the overlap ratio between the area of intersection $I_{P^qP^m}$ and the area of the query footprint $P^q$ and let

$$O^m = \frac{A(I_{P^qP^m})}{A(P^m)}$$

be the overlap ratio between the area of intersection $I_{P^qP^m}$ and the area of the footprint of the retrieved candidate building $H^m$ we only accept the placement if both overlaps $O^q$ and $O^m$ pass a user-defined threshold $\tau > 0.85$. Thus, a small amount is acceptable as the probability of finding a building with a perfect matching footprint is very small due to the relatively small size of the building database.

### 5.1.6 Synthesized Cityblock Layouts

In order to evaluate the city block layout transfer algorithm, we conducted several experiments that include reconstruction of a city block layout, exchanging footprint layout style by a different one, extending a partial city block layout, and synthesizing a city block layout for a street network generated by the algorithm described in section 4.2. We prepared a large set of example city blocks from European and US Cities, including: Augsburg, Bonn, Braunschweig, Bremen, Cologne, Frankfurt, Nürnberg, New York, Brooklyn, Queens, Huntington, that are used for different experiments.

**Reconstruction of a City Block Layout** In a first experiment, the goal is to reconstruct a city block layout of a part of the German city of Braunschweig. The purpose of this experiment is to show that if we are able to rebuild an existing layout, then the algorithm should produce meaningful results in virtual road networks, that contain city blocks that share similar properties as a real-world city block layout. We extracted the city blocks from the whole town of Braunschweig

Figure 5.6: The city block layout of a part of the German city of Braunschweig. This layout serves as a reference for the experiment in which this layout is going to be reconstructed using the city block layout transfer algorithm.

in order to conduct this experiment. The result of this extraction is 830 city blocks in total that might be used during the synthesis procedure. The part of Braunschweig we are going to use for this experiment consists of city blocks of varying shape and size and is illustrated in Figure 5.6. We used the algorithm in section 5.1.3 in order to match the individual blocks against the city block database of Braunschweig. The *Shape Contexts* were computed using 100 evenly spaced sample points that were sampled along the boundary of the city blocks. Each

shape context was constructed with eight angular bins and five distance bins. The final reconstruction is depicted in Figure 5.7 and it is identical when compared to the original layout. The result does not only show that the shape matching algorithm works correctly. It also shows that in case the richness of the database is large and thus contains city blocks that share similar or even identical properties as the city blocks used as query objects, this simple methodology that copies and pastes the layout already produces meaningful results.

Figure 5.7: Reconstruction of the layout shown in Figure 5.6. In order to achieve this reconstruction, a database containing all city blocks from the German city of Braunschweig was used.

The next experiment, we evaluate filling the same part of the city of Braun-

schweig; however, instead of using city blocks from Braunschweig, we use a collection of city blocks from different German cities: *Bonn, Bremen, Frankfurt, Munich, Nürnberg*. All city blocks from these cities result in a database of about 5000 exemplars. The result is shown in Figure 5.8 and it can be recognized that city blocks with a sparse building footprint layout look more plausible than city blocks that contain densely packed buildings, located near the street edges. In such cases, the shape difference between the query and the best match city block are still too large in order to successfully transfer the layout style. The eye-catching flaws of the transferred layout are the non-constant distance (increasing or decreasing) of the building footprint to the nearest street edge and corner buildings that might get culled away because they are partially located outside the query city block after alignment. However, the number of city blocks affected by such artifacts is typically very small. The overall layout looks plausible and realistic when the number of small blocks containing artifacts are neglected. Due to the independent matching strategy, it is conspicuous that in many cases the layout style is not coherent between neighboring city blocks.

**Exchanging the Building Footprint Layout Style** In this experiment, we exchange the city block layout of a part of the German City of Heidelberg with city blocks from different US cities. The goal of this experiment is to show that our algorithm can be used for layout style transfer and its result is illustrated in Figure 5.9. In order to conduct this experiment, we extracted city blocks from different parts of New York and Huntington. The size of the database comprises about 13500 downtown city blocks. When one analyzes the city block shape of US-cities it can be recognized that most city blocks have a rectangular or nearly rectangular shape. Other shapes such as triangular or trapezoid ones can be found occasionally; however, these shapes occur much less often. The chosen part of Heidelberg contains a significant amount of rectangular or nearly rectangular city block shapes, while only a few of them have a very irregular shape. The richness of the city block database is large enough to infill city blocks with a plausible and realistic footprint distribution. Even trapezoid shaped blocks are present within the database. In case of a densely packed footprint layout, even a nearly constant distance to the nearest street edges is preserved. Triangular and more irregular shapes are clearly 'underrepresented' within the city block database. The closest triangular shapes have been placed and produce artifacts such as non-constant distance to the street edges and corner buildings that were culled away. Especially, US-cities contain city blocks that have a strongly varying building density. Here it would be necessary to incorporate neighborhood information to produce a coherent layout across multiple city block boundaries.

Figure 5.8: A city block layout filled by re-using exemplar city blocks from German cities: Bonn, Bremen, Frankfurt, Munich, Nürnberg. For adjacent rectangular shaped city blocks, plausible and realistic layouts are achieved using the city block layout transfer algorithm. For irregularly shaped blocks the result strongly depends on the richness of the exemplars present in the database.

**Extension of a Partial City Block Layout** Another application of the city block layout transfer is extending an existing city layout. Figure 5.10 illustrates a part of New York near the administration boundary. Such regions where no building footprints have been mapped by the community still occur very often across different countries and cities. In the following experiment, we showcase the infill of these

Figure 5.9: In this image, the layout style of the city blocks from a part of the German city of Heidelberg is exchanged by city block layouts from different US cities. The richness of the city block database is large enough to infill city blocks with a plausible and realistic footprint distribution. Even trapezoid shaped blocks are present within the database.

empty regions and thus extend the city block layout of New York using city blocks from different US cities, i.e., New York (Queens, Brooklyn) and Huntington.

The synthesized layout is illustrated in Figure 5.11. The city blocks located on the left-hand side of the red line belong to the original layout, while the city blocks on the right-hand side of the red line. The database used to infill the missing city block layouts has a size of about 13500 city blocks that were taken from New York (Queens, Brooklyn) and Huntington. The query and the example blocks used in this experiment come from US-cities. Typically, city layouts from US-

Figure 5.10: City block layouts are not present in every region of different cities. This is exemplified by looking at the administrative boundary of New York. Outside the administration level, no building footprint layouts are present within the city blocks.

city centers tend to have rectangular or at least near rectangular shapes. This holds for the layout shown in Figure 5.11, where a large amount of city blocks is nearly rectangular. However, even in this layout, city blocks with more complex shapes such as l-shapes are present. For each of them, a suitable block in the database is present and the placed city block layouts look plausible and realistic. It is eye-catching that the density of the building footprint within the blocks varies between the different parts of the layout. When one takes a look on the left-hand side of the city layout, the density inside the city blocks is nearly constant up to a few exceptions. The varying building footprint density in our result is a direct consequence of the independent matching and insertion of the best candidate for each individual city block within the layout.

Figure 5.11: The goal of this experiment is to infill empty city blocks with example blocks that were taken from different US cities. Due to the richness of the database plausible footprint arrangements for near rectangular as well as complex irregular shaped city blocks can be synthesized.

**City Block Layouts for a Synthesized Street Network** In a final experiment, we fill a synthesized street graph with real-world city blocks (see Figure 5.12). In order to conduct this experiment, we synthesized a street network using the technique presented in section 4.2. As a database, we used city blocks from different German cities: Augsburg, Bonn, Bremen, Köln, Frankfurt, Munich, Nürnberg. The city blocks of the road network contain large shape variation and a substantial amount of irregularly shaped blocks, i.e., l-shape and even t-shape. In addition, the street network contains near rectangular city blocks that have wavy street segments between crossings. The footprint arrangements inside adjacent rectangular shaped city blocks look plausible especially in cases where the final layout is sparse, i.e., buildings are not densely packed side-by-side. However, the layouts within the irregularly shaped city blocks look less plausible, because the

Figure 5.12: A street network that was generated by StreetGAN (section 4.2) is filled with city blocks from various German cities. Near rectangular city blocks even with wavy street segments are filled with plausible footprint layouts. Due to a small variety of irregularly shaped city blocks within the database, the synthesized city blocks are less likely in irregularly shaped blocks, because several building footprints are removed that were partially located outside the city block polygon.

city block is not covered with building footprint along each of the street segments. This is a direct result from large shape difference between the individual query blocks and the best matching exemplar. The consequence is that different buildings, which are partially located outside the city block polygon, will not be transferred. In addition, the distance of the individual footprints increases or decreases along the street segments, a property seldom present in real-world footprint arrangements. Typically, the city block database contains much less variation of irregularly shaped blocks such as l-shaped ones in contrast to rectangular shaped city blocks that occur much more often. For such cases, the estimation of a rigid

transformation is not enough. Here, it would be necessary to compute the location of the buildings using a more elaborated algorithm which maintains the shape of the footprint as close as possible and aligns the building with it potential neighbors along the nearest road segment without introducing intersection between the building footprints.

### 5.1.7 Synthesized Building Layouts

In order to populate the synthesized footprint layouts with 3D buildings, we next show results using the technique presented in section 5.1.5. In order to evaluate the placement of buildings, we conducted several experiments. We downloaded 3D models from the public geometry repository *TrimbleWarehouse 3D* and categorized them into different groups such as 'skyscrapers' or 'residential' buildings. In this first experiment, we evaluate the placement of skyscrapers on the footprint layout of a part of the German city of Braunschweig. The building footprint within the original layout has a nearly rectangular or trapezoid shape, l-shaped footprint with different length of both flanks, and a few irregular shaped footprints that even contain curved regions. We used two different skyscraper data sets to conduct the experiment. One contains 400 skyscrapers from New York, and the other one contains 300 skyscrapers from Chicago. Figure 5.13 show a top view and Figure 5.14 shows a perspective view of the placed 3D buildings using the database of 400 building models from New York. In the top view (see Figure 5.13) it can be recognized that a building was placed for every footprint present within the original layout. This demonstrates that even with a small database of buildings at hand, it is possible to populate an area with buildings that have already been modeled for a different use case. The database is rich enough and provides building footprints of various shapes: nearly rectangular, l-shapes with different lengths of the flanks. It is eye-catching that several building footprints have nearly identical shapes. The matching strategy is done independently for each building footprint; thus, the same building might be placed multiple times within the same city layout.
Conducting the same experiment with a different database using 300 models from Chicago leads to a different result, which is illustrated in Figure 5.15 and Figure 5.16. As the database does not contain a large variety of buildings having an l-shaped building footprint, the overlap check discussed in section 5.1.5 fails in 14 % of the cases. As a consequence, no building will be placed, and the footprints stay empty.

### 5.1.8 Analysis and Comparison

**City Block Layouts** Existing methods for computing city block layouts rely on algorithms that divide the space enveloped by a city block using procedu-

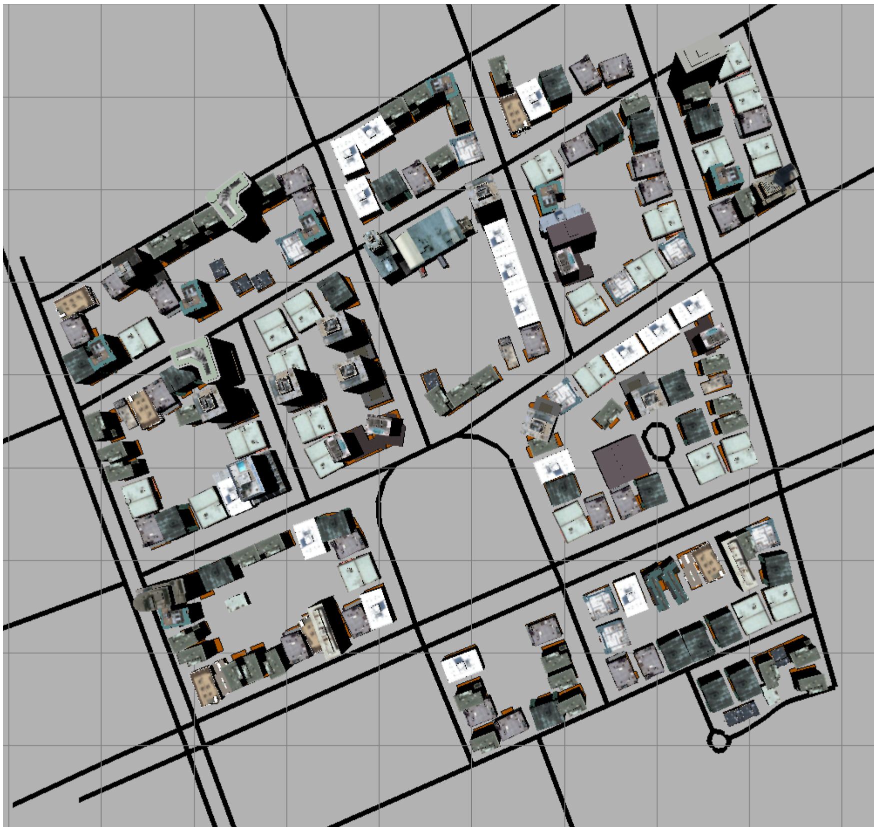


Figure 5.13: A footprint layout from the German city of Braunschweig. Each building footprint was populated with a 3D model having a similar sized and shaped building footprint. The overall size of the building database is 400 skyscrapers from New York.

ral or rule-based algorithms. Different subdivision schemes for city blocks have been proposed such as [VKW+12, PM01, AVB08]. In Yang et al. [YWVW13] procedural templates have been used to generate parcel layouts. In contrast to our approach, these algorithms are designed to produce parcels that provide the space for different buildings. The different building footprint shapes typically depend on the technique that generates the buildings, i.e., the shape grammar [WWSR03, MWH+06]. Our approach re-uses real-world exemplars that contain a huge amount of varying building footprint layouts. In principle, our source of data may carry parcel subdivisions; however, specific parcel layouts are very rare and are not present for the most of the city blocks.

**Placement of 3D Buildings** The standard way to populate buildings on predefined building footprints is to use procedural algorithms using shape grammars

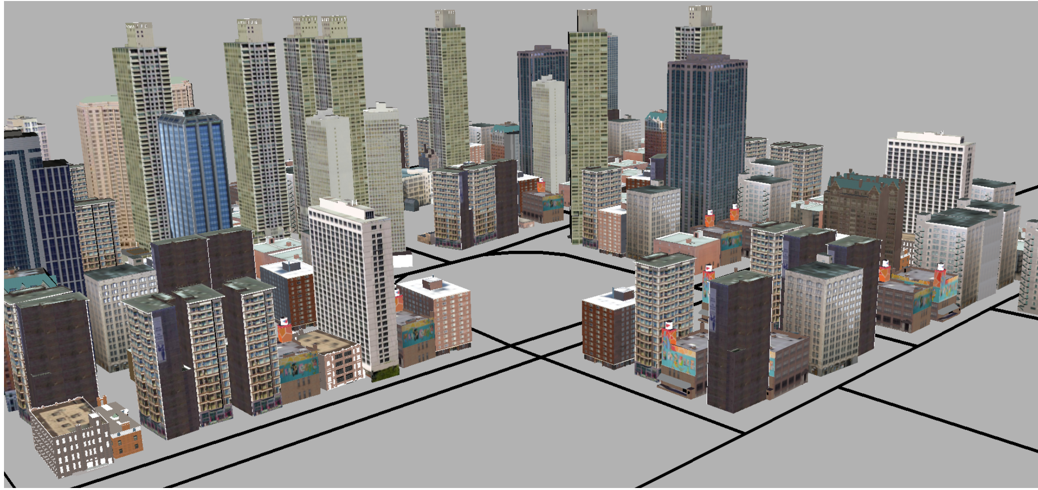


Figure 5.14: A perspective view onto the populated footprint layout from Figure 5.13.

[WWSR03, MWH+06]. However, the use of shape grammars is typically cumbersome and labor-intensive and has a very steep learning curve. For these reasons, we decided to follow the example-based methodology that is less labor-intensive as the exemplars can be easily downloaded from public geometry databases. However, the quality of the available 3D models strongly varies in both, level of detail and quality of the delivered textures. Especially the texture contains many unpleasant effects such as different light intensities, reflections, and cars and trees in front of the different buildings. Apart from this, our technique shows that plausible building layouts on top of existing footprint layouts can be synthesized even if the size of the database is small. Footprints that will not get covered with a building, due to large footprint shape difference can be treated using individual techniques to synthesize a building on the fly, e.g., a shape grammar [WWSR03, MWH+06] or an example-based building synthesis technique (see section 6).

### 5.1.9 Limitations

The example-based city block layout transfer produces city layouts that mimic the style of footprint arrangements found in the real world. Especially, with the additional placement of 3D buildings on top of the synthesized layout the realism of the layout is drastically increased. During the evaluation of the results presented in section 5.1.6, we identified several limitations that will be discussed next. The city block layout transfer algorithm has a significant dependency on the city block database. There exist cases where the database does not contain viable city blocks where the shape difference between the city block used as query object and the re-

Figure 5.15: The same building footprint layout shown in Figure 5.13. In this experiment, a different similar sized database was chosen. The result of the building placement strongly depends on the shape variety present within the footprint of the 3D building models. About 14 % of place buildings to do not pass the overlap check discussed in section 5.1.5.

trieved candidates is small. In such cases, the transferred building footprints might not be located along the enclosing street segments. One possible solution would be to increase the size of the city block database further. However, pursuing such a strategy will still not guarantee that every possible query shape is present within the database. The same problem occurs during the placement of 3D buildings on top of the synthesized building footprints. To alleviate such cases, we developed an algorithm that compensates the large shape difference by synthesizing a novel layout that preserves the overall style of the original building footprint layout. This algorithm will be presented in section 5.2.
The example city blocks that are retrieved as viable candidates are inserted independently of each other. This might lead to drastic layout changes between adjacent city blocks. However, there is currently no metric to compare the simi-

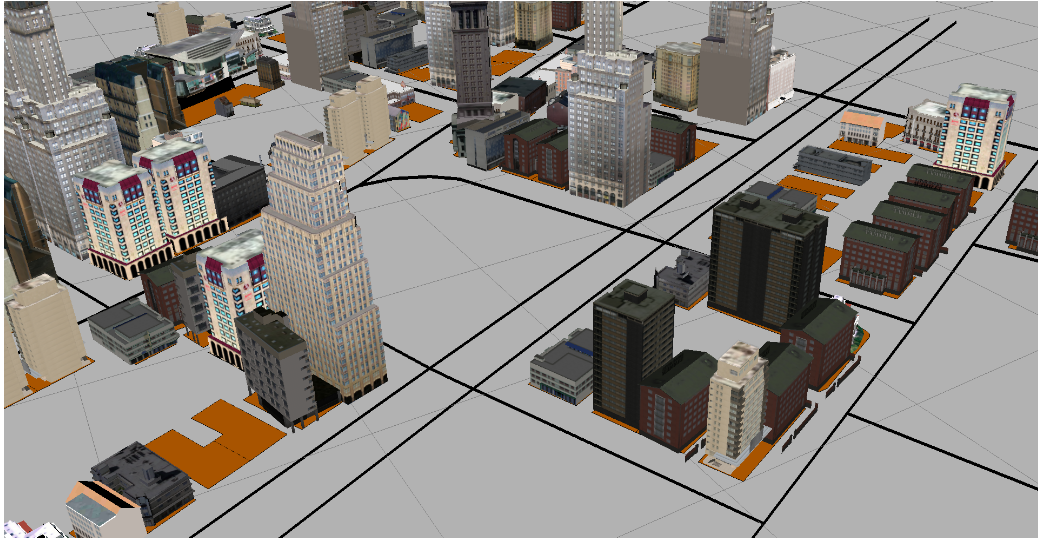


Figure 5.16: A perspective view onto the populated footprint layout from Figure 5.15. It can be recognized that for similar shape building footprint the same example building might be used several times.

larity of the building footprint arrangements within city blocks, especially when the shape and the size of the city blocks strongly differ. In contrast, to the algorithms that subdivide a city block into a set of parcels (see Section 2.2.1), our algorithm cannot produce or copy such parcels, because they are not present in the example city blocks. However, parcels are especially necessary to assign a specific area of land around a building. One possibility to overcome this lack of information within the example city blocks would be to reconstruct a cadastral topological map using *Voronoi* diagrams using the 2D outline or its segments as sites. This would lead to a coarse subdivision of the city blocks into a set of parcels enveloping each of the synthesized building footprints. To overcome such effects, the work at hand presents a specialized re-synthesis scheme custom tailored to city blocks in Chapter 5.2.

## 5.2 Guided Re-Synthesis of Discrete Element Arrangements

Herein, a novel re-synthesis scheme for layouts composed of sets of discrete elements is described. The proposed approach utilizes a guidance map that is content aware and can be used to synthesize a novel layout while maintaining the style of the original one. The novel re-synthesis scheme is evaluated in-depth, and several challenging re-synthesis results on real-world city-blocks are illustrated. The content of this section is based on the peer-reviewed publication

> Stefan Hartmann, Björn Krüger, and Reinhard Klein. Content-aware re-targeting of discrete element layouts. In *International Conference on Computer Graphics, Visualization and Computer Vision*, volume 23 of *WSCG proceedings*, pages 173–182, June 2015.

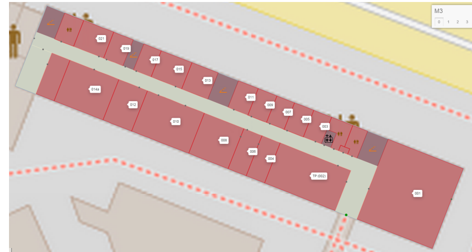
and contains text parts of the mentioned publication that were copied without further modification. These text parts are highlighted in gray.

### 5.2.1 Motivation

Road networks subdivide the urban space into smaller structures that are enclosed by roads. These structures are called city blocks. Naturally, the city block provides space for buildings and is typically subdivided into smaller regions, where the building footprints are placed. However, the road network generation algorithm only provides the road network. So far, no further structures such as building footprints are placed in inside these blocks. The primary goal of this chapter is to fill this empty space using an example-based technique. In contrast to section 5.1, where we studied techniques to transfer city block layouts using simple transformations, we focus here on large shape difference between a source and a target city block. As plenty of existing city blocks layout have been designed within cities all over the world, we have access to a large number of existing layouts. In order to transfer the real-world layouts to a target domain, we need to bend or re-structure the original layout to fit into the target city block. This might involve a continuous deformation to the original layout, thus distorting the original arrangements and even their discrete elements. Instead of deforming and repairing distortions, our goal is to re-synthesize a plausible layout that resembles the style of the original layout locally and even globally using discrete atomic building blocks. Our key insight for synthesizing such a new layout that mimics the style of the original one is to feed the synthesis technique with a guidance map, that encodes the style present in the initial layout. Thus it is content aware. Such a guidance map can be retrieved from a modification of the initial domain, such

as the appliance of a continuous deformation to the initial layout elements. Using such a guidance map during the synthesis has two major advantages: First, it allows our synthesis technique to resemble the global style of initial layout, and it even allows introducing discrete copies of the original elements to maximize the layout compliance, *i.e.* the local style. Second, it guides the algorithm during the exploration of the layout space and allows for efficient pruning this typically large search space. In contrast, to the procedural approaches that were discussed in detail in section 2.2.1, we focus on the synthesis of a novel layout based on an existing one. In terms of urban modeling, we want to compute a city block layout, that resembles the style of a real-world city block, e.g., taken from *OpenStreetMap*. The approach implicitly uses adjacency information from the guidance map to preserve the style, while methods such as Merrel *et al.* [Mer07, MM09] or Lin et al. [LCOZ+11], extract this information in a preprocessing step that sometimes even requires manual work. Layout synthesis algorithms based on Markov Chain Monte Carlo (MCMC) are also heavily used for the computation rooms or floor plans [YYT+11, YYW+12, YBY+13, MSL+11]. However, these methods use predefined or learned object relationships in order to determine the probability or fitness of a current layout state. These types of algorithms are typically computational expensive, and the produced results strongly depend on the number of iterations used for the optimization procedure.

(a) A real-world city block with discrete building footprints taken from a block within the city of Bonn.



(b) An office floor plan composed out of a set of discrete room instances.

Figure 5.17: Real world layout examples that both consist of a set of discrete elements. Both examples are taken from *OpenStreetMap*[Ope17a].

### 5.2.2 Re-synthesis Domain and Element Arrangements

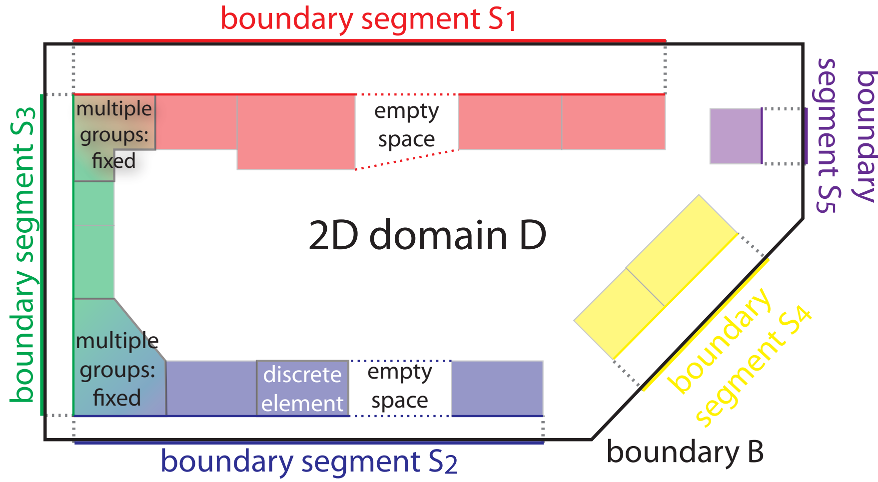



Figure 5.18: Abstraction of a complex scene as 2D domain $D$ with a set of discrete elements. The domain is enveloped by a boundary $B$ split into subregions $S_i$, each assigned with a set of discrete elements.

A large variety of real-world structures can be abstracted as a set of polygons enveloped by a closed boundary. Typical examples for such real-world structures are building footprints located inside a city block or rooms located in a floor plan (Figure 5.17). We define $\mathscr{P} = \{P_1, \ldots, P_n\}$ as the set of $n$ polygons, each representing a 2D outline of discrete elements, placed within a complex scene. Each of them may carry a set of application specific annotations. These elements located in a scene are typically enveloped by a closed curve denoting its boundary $B \in \mathbb{R}^2$. The area enclosed by $B$ represents the scene itself and is a planar domain $D \in \mathbb{R}^2$. Inside this area, we expect that the elements are arranged into a set of $m$ groups $\mathscr{G} = \{G_1, \ldots, G_m\}$, each containing a subset of the discrete elements (Figure 5.18). However, the initial layout does not necessarily have to be a tightly packed one, *i.e.*, elements need not directly placed next to each other, there might be empty space in between them.

Before we state the problem definition, we first analyze the atomic entities within the domain $D$ and merge them into a set of arrangement groups $G$. In our setting, we expect that the discrete elements are located within an application specific distance near the boundary $B$ of the domain $D$. The boundary might be further split into a set of boundary segments, separating the initial boundary into a set of $l$ smaller entities $B = \{S_1, \ldots, S_l\}$. The segments $S_j \subset B$ typically represent a road segment which is part of a city block boundary or a wall inside a building or room. In order to merge the elements into groups, they need to be assigned to the segments $S_j$ along the boundary $B$. In order to decide which element is assigned to a certain group, we determine the Hausdorff distance $h(P_i, S_j) \forall S_j \in B$

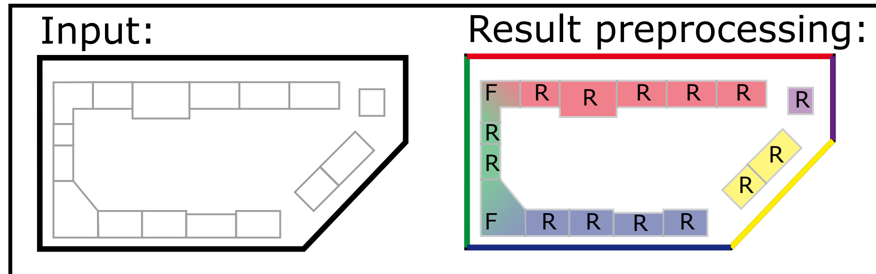



Figure 5.19: As preprocessing step, the domain $D$ is analyzed, and its discrete elements are assigned to arrangement groups $G$ illustrated by the different shaded colors. Additionally, each discrete element is attached a behavioral type *fixed*(F) or *repeatable*(R).

between the outline of the discrete element $P_i$ and the boundary segments $S_j \subset B$. In case $h(P_i, S_j) \leq \tau$ falls below an application specific threshold ($\tau = 25.0$ meters in our city block application), the discrete element instance is assigned to that certain arrangement group $G_j$ that corresponds to a boundary segment $S_j$. It might happen that a few elements will be assigned to multiple boundary segments, this is an indicator that the element is located at the end of an arrangement group and might need special, application specific, treatment (Figure 5.18). Most of the elements are typically assigned to at least one boundary segment $S_j$. The count of assignments of a discrete element $P_i$ to different $S_j$ is used to deduce three behavioral attributes, *i.e.* the element type, for the element instances:

- *fixed* (F): elements with two or more assignments, are not allowed to be repeated. These are elements placed at a street corner.
- *repeatable* (R): elements with a single assignment, allowed to be placed multiple times. An original element might be stretched within the guidance map. In order to compensate for increased space usage, we repeat the element multiple times.
- *empty space* (E): a special element that serves as a fill region, preferably placed in regions, where no discrete element covers the space within the group $G_j$. The purpose of this element is to discretize empty space, i.e. space where no element is placed. In order to reproduce empty space in the new layout, we assign such an element to each group $G_j$. As it is a virtual element that will not be visualized in the result, we use a fixed dimension of $w \times d = 1.0\, m \times 10\, m$.

### 5.2.3 An Efficient Algorithm for Interactive Re-Synthesis of Element Arrangements

#### Deformation as guidance information

Re-synthesis of an existing layout is necessary if the different operations are applied to the initial domain $D$. The editing operation might include the movement of a boundary vertex or the relocation of a street segment $S_j$. Another application would be the insertion of an example city block into an empty city block located inside a virtual street network. Such an application was already proposed in section 5.1. Both applications imply that the interior of the city block or domain is affected by applying a map $\phi : D_{in} \to D_d$, where $D_{in}$ denotes the initial domain that and $D_d$ denotes the deformed one. Computing such mappings is well researched and can be achieved by employing different techniques such as the well-known *Mean Value Coordinates* [HF06] or (quasi)-conformal mappings [LKF12]. In this work the *Mean Value Coordinates* were used to realize the mapping $D_{in} \to D_d$. When $\phi$ is directly applied to the discrete elements and their corresponding geometry located within the domain, visually unappealing artifacts might be the consequence *i.e.* the elements get distorted and unnaturally stretched/warped.

One possibility to avoid deformation artifacts would be treating the objects as single points, represented by their centroid. This might avoid object deformations; however, will usually lead to other artifacts such as element overlaps and dissolved element groups, *i.e.* element get overlapped with other elements or elements that were exactly placed next to each other will set apart in the new layout. Such effects might be reduced by applying smarter deformation techniques; however, this claims for additional knowledge about the objects' structure, which is not the scope of this work. We follow a different line here and compute a novel layout out of the original elements using a deformed version of the original layout, which we call *guidance map*. The guidance map is a set of deformed discrete elements $\mathscr{P}'$ that are the direct result from applying the map $\phi$ to the interior, *i.e.* the discrete elements $\mathscr{P}$ of the initial domain $\phi : \mathscr{P} \to \mathscr{P}'$. Although the elements are deformed, the layout still contains important information about the original layout. It includes groups, *i.e.* elements that are placed next to each other in the original layout $\mathscr{P}$ will be placed next to each other in the deformed layout $\mathscr{P}'$. Further, the presence of empty space and the ordering of the discrete elements is preserved $\mathscr{P}'$. We use this valuable information encoded in the set of deformed discrete elements $\mathscr{P}'$ to compute a novel layout that mimics the style of the original layout.

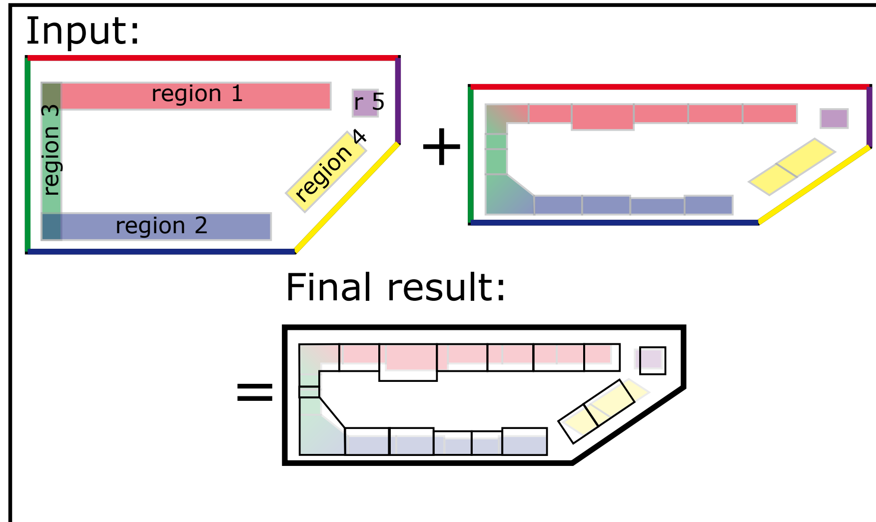



Figure 5.20: When interactively editing the domain, deformations are applied to the boundary. The interior of the domain is deformed and serves as prior to our iterative synthesis technique, that computes a novel layout guided by the deformed discrete elements.

### Re-Synthesis Formulation using 1D Sequences

In the following the solution for synthesis of a novel layout that preserves the properties of the initial one is presented. Before we show the mathematical formulation to tackle the problem, we comprehensibly explain the styles that can be maintained by the proposed algorithm. We focus on the following three properties to achieve our goals:

1. Element groupings present in the initial domain shall be preserved. This means elements that are placed exactly next to each other in $\mathscr{P}$ will be placed exactly next to each other in $\mathscr{P}'$.

2. Elements prefer to be located at similar relative positions, and

3. Elements prefer their original neighborhood.

In our setting, the term neighborhood does not express the local similarity of geometric attributes. Our goal is that we simply want to preserve the local element ordering found in the initial domain. Up to now, we have discussed the input data and its segmentation into a set of discrete elements and their grouping into arrangement groups $G_j$. From now on, we focus on formulating the re-targeting

problem as an optimization problem which can be solved efficiently, thus allowing to explore solutions for different domain deformations interactively.

**Problem Formulation** The key idea for re-synthesizing a deformed domain is its formulation as a labeling problem. This type of problem formulations is well studied in the computer vision community, and several efficient solution techniques exist [BVZ01]. While the formulation of a labeling problem is typically done for 2D problems *e.g.* image segmentation, we exploit the sub-structures found in our problem setting, namely the linear element arrangement groups. This allows us to solve the layout problem employing efficient algorithms. These considerations result in the following equation to be minimized:

$$L(f_1,\ldots,f_N) = \sum_{k=1}^{N} O(f_k) + \sum_{k=1}^{N-1} \delta(f_k, f_{k+1}) \tag{5.1}$$

The $f_k$'s are representations of discrete elements that might get placed at different discrete positions located within the parameter domain of the boundary segments $S_j$. Thus, a $f_k = (P_i, m, d)$ is a tuple that consists of a reference to a discrete element $P_i$, its current discrete position $m$ and $d$ being the length of $P_i$ projected onto the boundary segment $S_j$. In our setting the $d$'s are rounded to be a multiple of 1.0 *meter*. The energy we want to minimize in order to find an optimal sequence of discrete elements is given in Equation 5.1. This energy is composed of two terms, that can be mapped to our notion of style preservation outlined above. The data term $O(f_k)$ measures the cost for assigning an element $f_k$ to a specific location $m$. Thus, it penalizes elements that will be placed at positions, where they do not overlap with their corresponding distorted instance $P_i^{'}$ within the guidance map. From a different view, it can also be seen as prior term pulling the original elements towards the position, where the deformed instance of the particular discrete element is located. We designed $O(f_k)$ to be the area difference between the original element $P_i$ and the area of the intersection of both original $P_i$ and deformed $P_i^{'}$ defined as follows:

$$O(f_k) = A(P_i) - A(P_i \cap P_i^{'})$$

Thus $O(f_k)$ penalizes space not covered between its discrete element $P_i$ and its corresponding deformed $P_i^{'}$. Further, it depends on the current location $m$, where $f_k$ might be positioned inside the result sequence, and it changes every time the domain is modified. The transition between two consecutive discrete elements is penalized by $\delta(f_k, f_{k+1})$ that is defined as follows

$$\delta(f_k, f_{k+1}) = \begin{cases} \alpha & \text{if } f_k \in E \wedge f_{k+1} \in E \\ (1-\alpha) & \text{if } f_k = f_{k+1} \\ 0 & \text{if } f_k \neq f_{k+1} \end{cases}$$

This function encourages omitting two identical discrete elements $P_i$ getting placed next to each other and instead prefers different instances. In order to achieve continuous regions of empty space, when present, we penalize two consecutive empty space elements less severe, than two identical $P_i$. In our examples we used $\alpha = 0.05$. As described in Section 5.2.2 we exploit that, elements can be grouped into arrangements along segments of the boundary polygon. For each of them, the optimization problem boils down to computing a sequence of elements along a curve, that in total fulfills a certain minimal length $M$ along the parameter domain. In order to employ an efficient algorithm, that minimizes Equation 5.1 the $f_k$'s are restricted to be placed at discrete positions along the curve. The discretization depends on the specific application (we used discrete steps of 1 meter in our examples). This allows us to reformulate the objective function in the following, recursive way:

$$\begin{aligned} L[f_1, m_{f_1}] &= O(f_1) \\ L[f_k, m_{f_k}] &= O(f_k) + \min(L[f_{k-1}, m_{f_{k-1}}] + \delta(f_{k-1}, f_k)) \end{aligned} \tag{5.2}$$

We solve this recursive formulation using a graph-based dynamic programming approach adapted from Lefebvre *et al.* [LHL10]. In our setting we need to incorporate both, node costs $O$, *i.e.*, overlapping cost and neighbor costs $\delta$, *i.e.*, 'concatenation costs' when growing the graph implicitly. We can terminate the growing in case if the sequence length $m$ including the current top node of the queue is $m \geq M$. During the graph expansion step, the paths are managed by a priority queue, where the least cost path can be extracted by extracting the node at the front of the priority queue.

$$\underset{m^* \geq M}{f_k^*} = \text{argmin}(L[f_k, m_{f_k}] + \delta(f_{k-1}^*, f_k)) \tag{5.3}$$

If the condition $m \geq M$ is satisfied the node that represents the front of the priority queue can be used to extract the optimal sequence $f_k^*$ (see [LHL10] for details). This is done by tracing back the parents until the start node *i.e.* the node, that envelopes the element that starts the sequence, is reached. The optimal sequence represents a sequence of discrete elements that best approximates the original layout along each of the $S_j$.

### Iterative Synthesis of Arrangement Groups

While the proposed formulation in Section 5.2.3 only guarantees a global optimal solution in case the initial layout consists of a single arrangement group located within the initial domain, we employ a meaningful heuristic in case multiple arrangements groups exist. Instead of combining and linking them together and

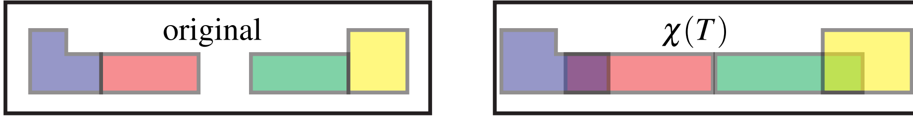



Figure 5.21: Concept modified guidance map. Left: the original map. Right: The guidance map is transformed by applying anisotropic scaling to the individual elements. Due to this deformation, the elements in the guidance map may overlap partially. This allows for a less restrictive positioning of the elements in larger areas.

optimize for a globally optimal layout, we have decided to choose an iterative algorithm, that synthesizes each arrangement group separately. This dramatically improves the performance of the synthesis procedure and allows to use it in interactive editing sessions. We decided to sort the groups by their cardinality of the discrete elements assigned to them. Thus, the synthesis technique starts with the most dominant group located inside the initial layout. The most dominant group is the one that contains the largest amount of building footprints within the original layout. In order to avoid overlaps with already synthesized sequences, we constrain the $j-th$ arrangement group by the previous $k \in [j-1,\dots,1]$ that potentially placed elements (fixed, repeatable) will not penetrate the convex hull of existing sequences.

### Different Guidance Strategies

As mentioned above the guidance map $\mathscr{P}'$ is the result of applying the map $\phi$ to the interior elements of the initial domain $\mathscr{P}$. Typically, the creation of such a guidance map is not restricted to a specific deformation method. It is possible to modify the size of the discrete elements in advance or apply a deformation scheme, where weights can be distributed using a simple brush metaphor. Such a scaling scheme was presented in the work of Möser *et al.* [MDWK08]. When using the guidance map $\mathscr{P}'$, the algorithm might present solutions, that contain multiple consecutive occurrences of discrete elements. This results from the guidance map, which encourages the algorithm placing elements in regions, where they overlap with their corresponding distorted instance, is cheaper than placing them somewhere else. This may result in visually unappealing repetitions, *i.e.*, the same elements is repeated multiple times (Figure 5.31(a)).
In order to overcome this unappealing side effect we use a modified version of the guidance map $\chi(T)$. Instead of applying the map to the initial discrete elements directly, we apply a scaling operator $T$ to each $P_i$ before applying the deformation map, thus $\chi(T) : \phi(T(P_i)) \rightarrow P_i^{''}$. As a result of this scaling, the deformed discrete

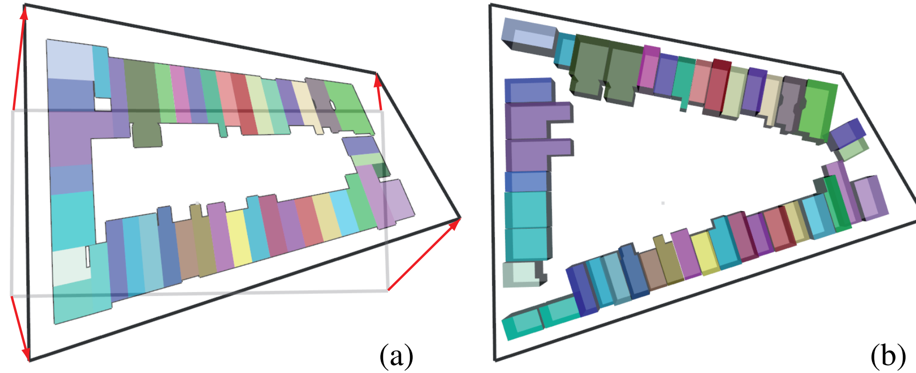



Figure 5.22: Re-targeting of a tightly packed city block. (a) Deformations (red arrows) applied to the initial domain (grey boundary) result in the edited domain (black boundary), distorted elements (various colors) serve as a guidance map. (b) Re-synthesized block layout.

elements now partially overlap and, therefore, the algorithm is able to bypass the original ordering temporarily. In our examples, we use anisotropic scaling along the direction of the boundary segment $S_j$ by a factor of 1.5 and apply it to each discrete element present in the corresponding arrangement group $G_j$. An example for such a modified guidance map $\chi(T)$ is shown in Figure 5.21.

### 5.2.4 Application: City Block Re-Synthesis

In order to showcase the versatility of our re-targeting approach, we evaluated our algorithm on a set of challenging test cases. Our primary application is the re-targeting of real-world city blocks. We extracted a set of city blocks from the well-known community mapping service *OpenStreetMap* (OSM) [Ope17a]. When we speak of a city block, we strictly speak of a simply connected and piecewise-linear closed boundary labeled as *street*. Inside, such a city block a set of polygons is located that are labeled as buildings in our examples.
Our first experiment, tackles the deformation of tightly packed city blocks, meaning that buildings and their 2-dimensional footprints are densely placed along the street segments.
In Figure 5.22(a) the deformation applied boundary intersections is highlighted by the red arrows. Applying this deformation to the initial layout results in distortions heavily shearing the elements and changing their size. The block re-layout computed by our algorithm (Figure 5.22(b)), resembles the style of the original block as a tightly packed layout is re-computed.
Figure 5.23 and 5.24 show different deformations applied to the same initial city

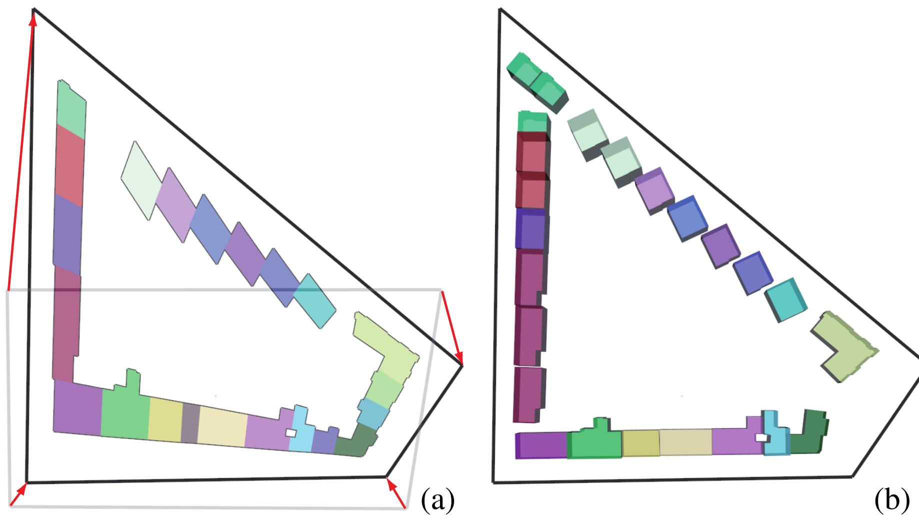



Figure 5.23: (a) Heavy enlargement of a city block. The applied deformation destroys the noticeable structure of diamond-shaped buildings. (b) Re-synthesized block layout: the style of the diamond-shaped buildings is preserved, due to original copies and additional elements are spawned as a result of the heavily enlarged edge.

block. Again, the red arrows highlight the applied deformations. In Figure 5.23 the block was heavily enlarged, in Figure 5.24 the blocks was moderately shrunk. In both cases, the noticeable structure and the orientation of the diamond-shaped buildings are preserved. The result further illustrates that if boundary regions are heavily shrunk or enlarged the algorithm is able to reduce the number of discrete elements or add additional ones.
In Figure 5.25(b) shows a re-synthesis of the initial layout of a city block shown

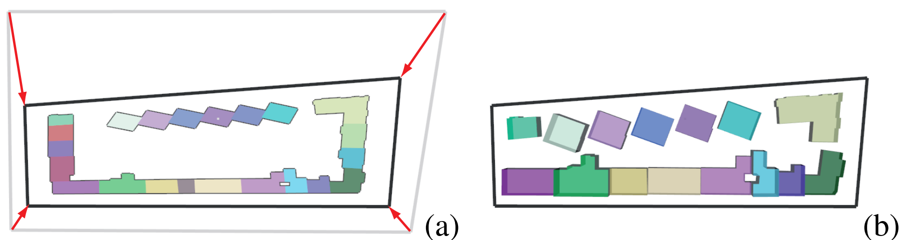



Figure 5.24: A city block that is moderately shrunk: notice, that elements get discarded from the layout in case the boundary regions reduced their size. However, the overall layout style is still recognizable.

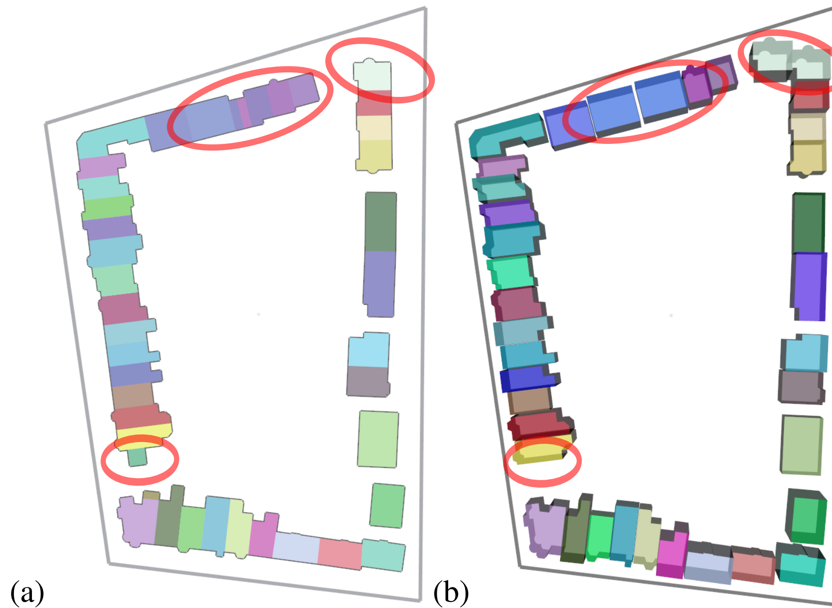



Figure 5.25: Re-synthesis of an original domain. The original layout (a) is used to guide the algorithm in order to synthesize the novel layout (b) into the original domain.

in Figure 5.25(a). Notice, the similarity between the initial layout shown in Figure 5.25(a) and the synthesized layout Figure 5.25(b). The algorithm is not able to reproduce the exact layout, due to the discrete nature of our approach. Figure 5.26 shows a result, where the outline of the original city block (Figure 5.25) was topologically modified by adding two additional intersections, leading to strongly visible shearing and bending of the elements within the guidance map. Using this guidance map as prior information for the layout algorithm, we are able to produce a plausible new layout, although some footprints present in the initial layout were discarded.

Other, examples demonstrating the strength our method are sparsely packed city blocks. Figure 5.27, illustrates that if empty space is present between buildings (five at the top street), the algorithm discards first empty space rather than discarding whole buildings since this would result in higher layout costs. For the result shown in Figure 5.28 we inserted two additional intersections along the edge with densely packed buildings. Notice that along the segment, where the split was introduced, the style (tightly packed footprints) is preserved, and even the space between the five buildings (top segment) is preserved in the synthesized result.

Figure 5.29 presents an additional result, which combines changing the topology of the boundary and applying a heavy deformation to the initial domain. Even in

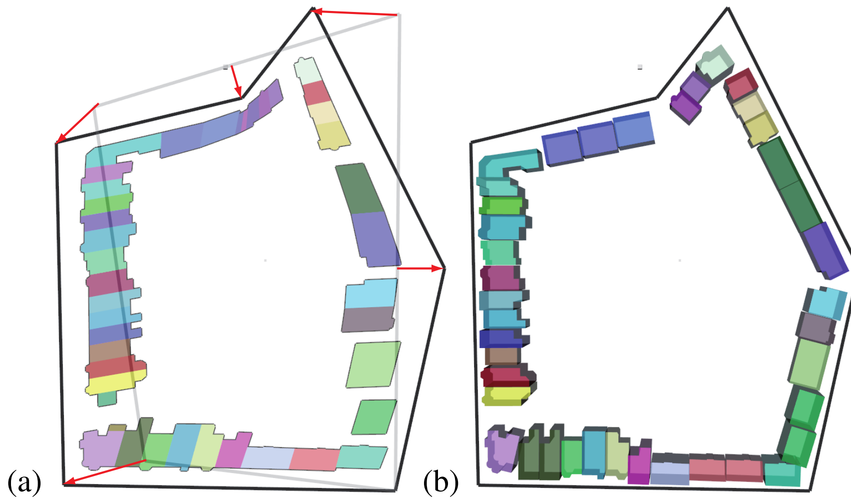



Figure 5.26: Topological modification of the city block outline, where two additional points were inserted. (a) Applied transformations and warped interior of the layout shown in Figure 5.25 (a). (b) Resulting layout: Our algorithm splits up the sequences at additional inserted points. Thus, the bending of the buildings is avoided.

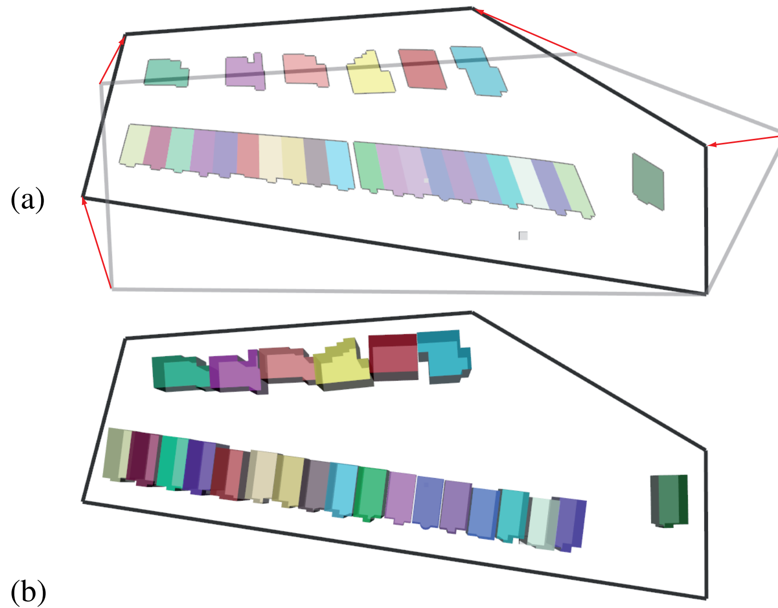



Figure 5.27: Sparsely packed city block, if edges get shrunk, and empty space is present in the layout our algorithm first discards empty space rather than discarding buildings.

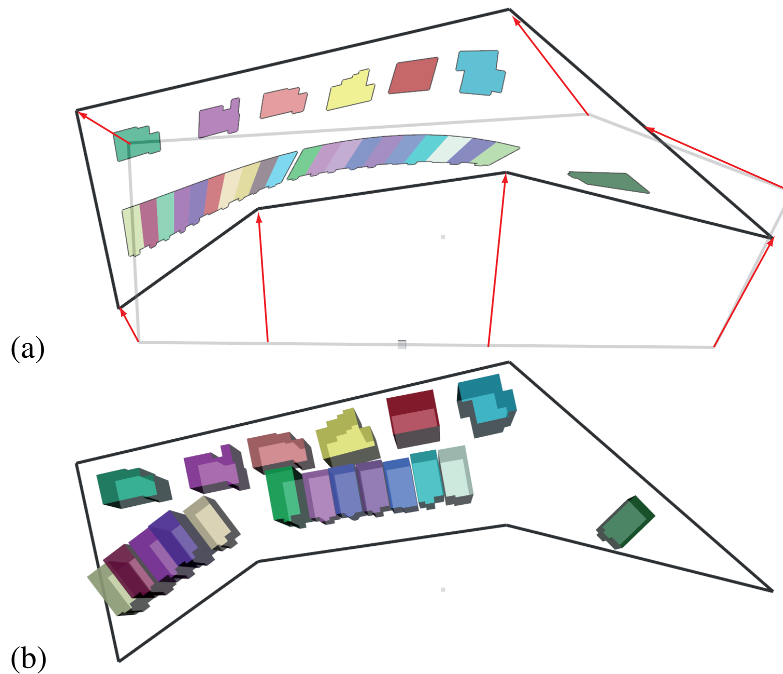



Figure 5.28: Deformed block after two additional intersections were added to the boundary. It can be noticed, that the continuous tight building layout is split into disjoint sequences, which are still tightly packed. Further note that space is preserved between the upper five buildings.

this in the case of a totally distorted domain, plausible building arrangements can be still synthesized.

In Figure 5.30, we present a result, where our approach was employed to a modeled city block populated with buildings taken from *Trimble Warehouse3d*. Please note, how additional buildings along the enlarged edges are introduced, and even the single tree present in the original block is replicated.

Finally, Figure 5.31 illustrates the usage of the modified guidance map. Notice that when an unmodified guidance map is employed, multiple consecutive repetitions are present in the result Figure 5.31(a) and (c). Employing a simple anisotropic scaling to the individual elements present within the guidance map reduces, the consecutive repetitions (Figure 5.31(b) and (d)). Finally, we want to note that generating the results found inside the paper took no longer than $0.15s$, using a desktop workstation with *Core i7 4930K (3.4 GHz)* and *32GB RAM*.

### 5.2.5 Analysis and Comparison

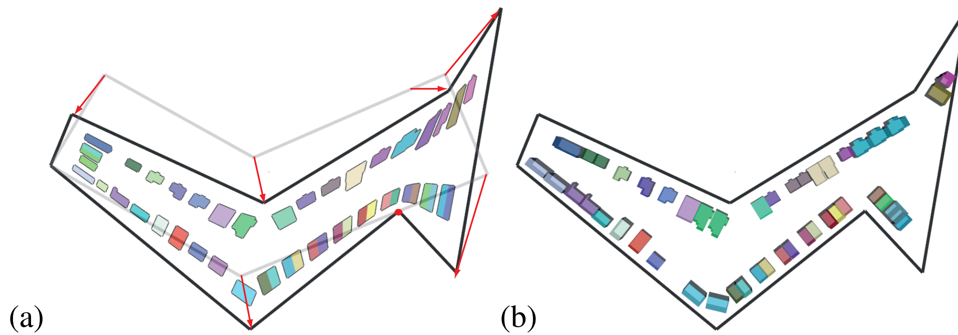



Figure 5.29: Heavily distorted block with additional intersections added to the boundary (a). Please note, that even applying such heavy deformations to the initial boundary, the element grouping (consecutive groups of two buildings) are present in the result (b).

So far, we presented a large set of different re-targeting results that were applied to several real-world city blocks. These contained several significantly different initial building footprint layouts. The city block portfolio used for the detailed evaluation of the presented re-synthesis scheme ranged from tightly packed city blocks, commonly found in downtown areas, to sparse suburban layouts containing building footprints with empty space in-between. The restriction to only generating layouts along the boundary segments allowed us to use an efficient graph search technique to synthesize a novel layout re-using the original discrete elements, i.e., building footprints. Compared to parcel generation algorithms [AVB08, VKW$^{+}$12, VABW09a] our algorithm enables the transfer of semantics attached to the real world footprints to the novel layout. This information could be exploited by an algorithm that places 3D buildings onto of these footprints such as the one proposed in section 5.1.5 and would enable more fine-grained constraints on the building types.

The presented algorithm might not be suitable for the synthesis of city block layouts with strong irregularities. Examples might be city blocks, where only a few large buildings are present, such as a school building next to office buildings, a sports facility, or a park. In such cases, it does not make sense to replicate the school building or the park instance. Typically, when such a block is placed the existing layout needs to be transferred as a whole by a geometric adaption of the footprint shapes.

## 5.2.6 Limitations

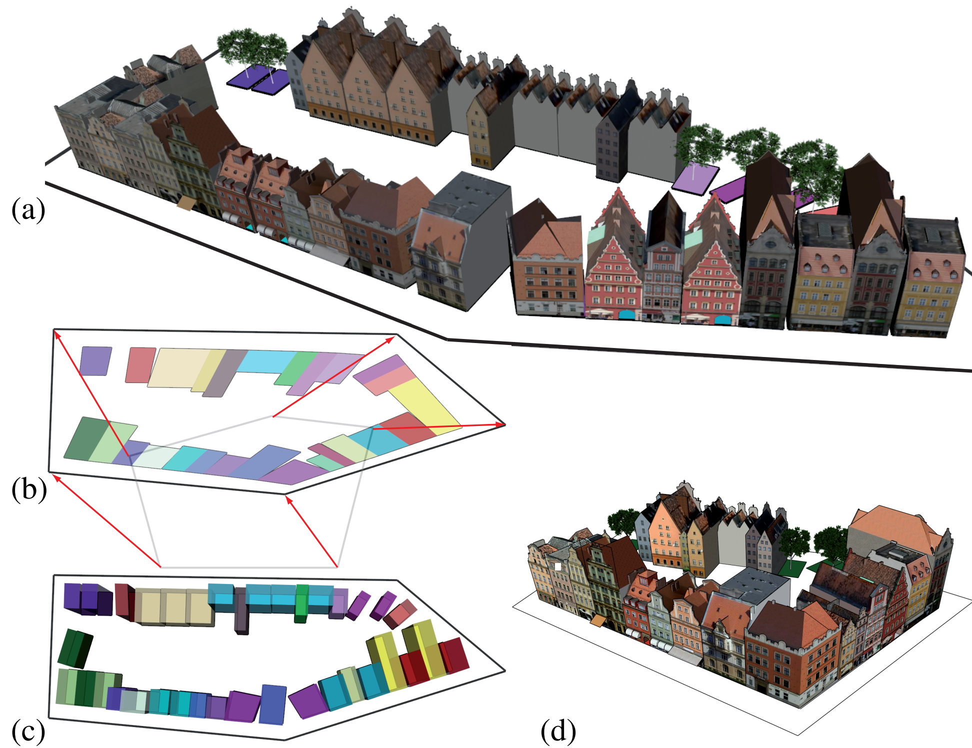



Figure 5.30: (a) A deformation of a city block populated with detailed modeled buildings. Notice, that even additional trees are inserted. (b) We show the deformation and the guidance map. (c) Top-view of the resulting layout. (d) The original building block.

Although, all these results look plausible, we identified a few limitations that need to be discussed in here. In our current implementation we rely on two conditions found in the input data (1) the atomic discrete elements blocks can be combined into larger groups, and (2) the elements are dominantly arranged along the boundary. Our algorithm is not restricted to element groups located near the boundary. In principle, any group of elements that dominantly follow a curve can be synthesized with our algorithm. Only a few modifications need to be realized to handle this case. (1) a method to fit a plausible curve, along the elements will be synthesized and (2) an additional data term, that handles, how 'well' the elements are oriented to the curve locally. Even in this case, we could again exploit the structure of the problem and still have an efficient algorithm to compute solutions. However, if no such groups are identifiable, our algorithm will produce failure cases. Further, we may note, that we currently do not see a promising straightforward extension of our approach for 'real' 2D dimensional domains.

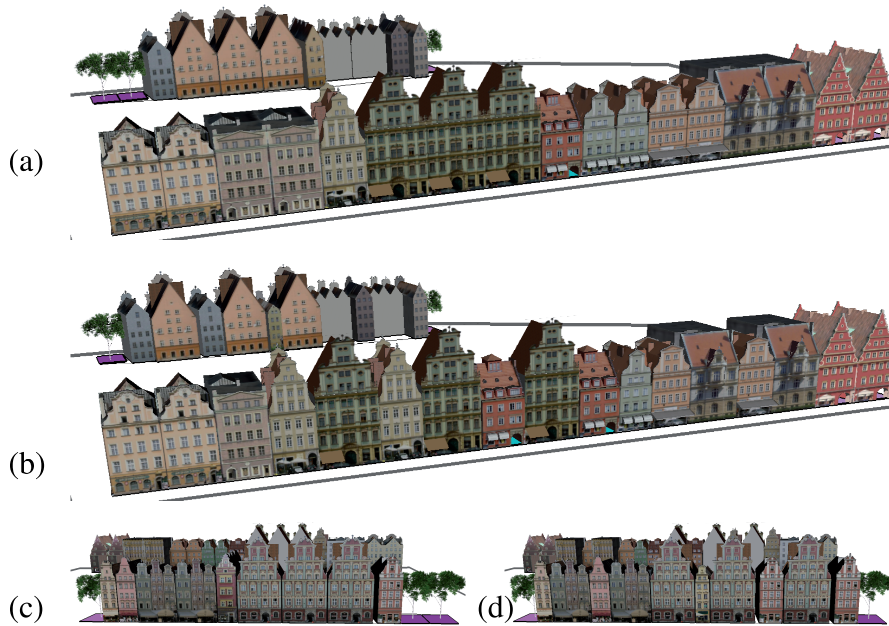



Figure 5.31: Illustrating the effect of using modified guidance maps. We show additional modifications to the block introduced in Figure 5.30. The results that are shown in (a) and (c) where obtained by simple warping of the guidance map. In this case, multiple consecutive repetitions of the same building are visible. To create the results that are shown in (b) and (d), simple an-isotropic scaling (referred to as blurring in Figure 5.21) was employed. This simple modification already reduced the number of consecutive repetitions significantly.

In addition, we identified another limitation introduced by our continuous deformation method acting globally across the polygon: It happens that even if the boundary edges do not change their length, the layout changes along these edges because the underlying guidance map has changed. This effect might be avoided by choosing a different deformation method, that can locally control the allowed deformation. We plan to investigate the approach presented by Möser *et al.* [MDWK08] for further evaluation.

# CHAPTER 6

# Example-based Building Synthesis

## 6.1 Motivation

Many types of content can be represented as sequences, such as text, 2D or 3D curves, audio data, videos, or media play-lists. Even content that is primarily used for urban modeling such as architectural textures [LHL10] and buildings can be represented as sequences of image strips or building parts. A sequential representation of a building can be achieved by cutting the initial building model into different slices along one primary dimension. A more abstract real-life example is the construction of building walls made of concrete wall elements. These wall elements are arranged along each other in order to form the different building sides. Sometimes they include even window and door openings. When this setting is transferred to the domain of example-based building synthesis, a novel building can be constructed by recombining individual building parts along one primary dimension. Typically, the envisioned 3D building model should satisfy a set of user-defined restrictions or constraints. Such constraints might be a certain length, a certain base area, a specific number of windows, or a certain area of solar panels placed on the building's rooftop.

An intuitive modeling metaphor for the synthesis of buildings might be derived by letting the user specify the shape of the building and a set of additional requirements. Thus, a building might be described by a skeleton shape and a set of $n$ requirements that demand a set of resources that need to be allocated in order to satisfy them. In this chapter, we propose to solve the synthesis of buildings as a generalized resource-constrained $k$ shortest path (RCKSP) problem, which was described in chapter 3.

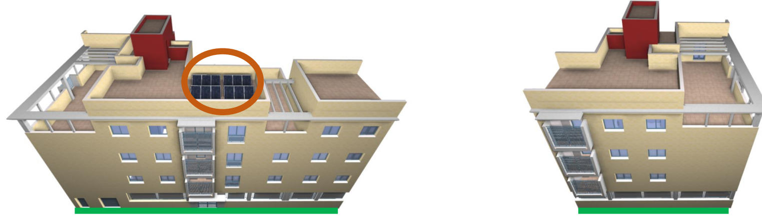


Figure 6.1: Building parts labeled as end-parts. On the left-hand side, a left-end part is illustrated, while the part on the right-hand side a right end part is illustrated. Both parts carry attributes such as length highlighted by the light green bar and other attributes such as the number of solar panels, number of air conditions or the presence of a front door.

## 6.2 Building Database and Annotations

The databases that are used for the building synthesis algorithm contain a set of building parts that manually prepared by segmenting existing 3D buildings from Trimble Warehouse 3D [Tri17]. All parts are labeled as left and right *end parts* (see Figure 6.1), *corners* with different angles (see Figure 6.2), *middle/filling parts* (see Figure 6.3) and *T-shaped parts* (see Figure 6.4), and annotated with different real-valued, integer attributes such as length, number of balconies, number of windows, number of doors etc. In addition, each part stores a polygon which represents the cross-section of the axis aligned cut for each side of the building part (see the blue polygon highlighted in Figure 6.2 and 6.4). The geometry of the building parts is scaled to quantize their length attribute (see the light green bars in Figure 6.1, 6.2, 6.3, 6.4) to multiples of 0.2 m. The transition cost between two building parts is defined as

$$\delta(e_i, e_{i+1}) = |A_{i,r} - A_{j,l}|, \tag{6.1}$$

*i.e.*, the area of cross-section disagreement between two adjacent parts, a measure of how well they fit together (see Eq. 6.1). The areas $A_{i,r}$ and $A_{j,l}$ represent the right cross-section of element $e_i$ and the left cross-section of element $e_j$, respectively. In all cases, we arrange the neighboring costs that are computed in advance and use a lookup table that is stored along with the database.

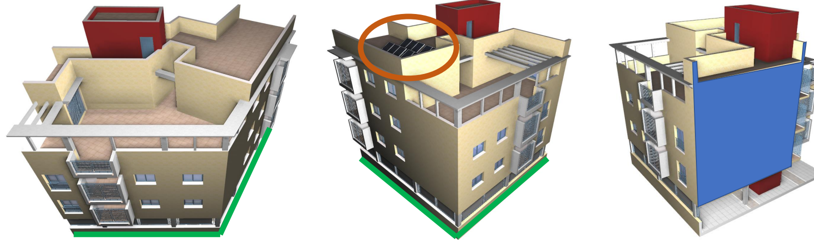


Figure 6.2: Building parts labeled as l-parts. On the left-hand side, two individual l-parts are illustrated both having different angles. The length attribute (light green bar) and other attributes such as solar panels are depicted by the orange ellipse. On the right-hand side, the cross section polygon of one side of the l-part is illustrated by the blue polygon.

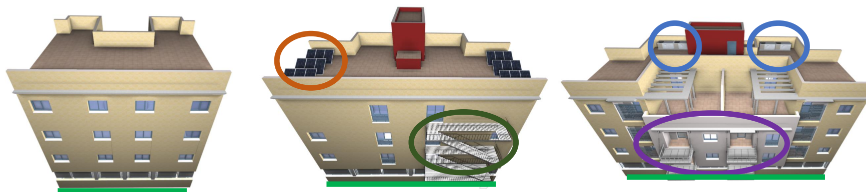


Figure 6.3: Variants of different building parts labeled as middle/filling parts. Building part attributes such as length depicted by the light green bar, solar panels, fire-ladders, balconies and air-conditions highlighted by the orange, dark-green, violet and blue ellipses.

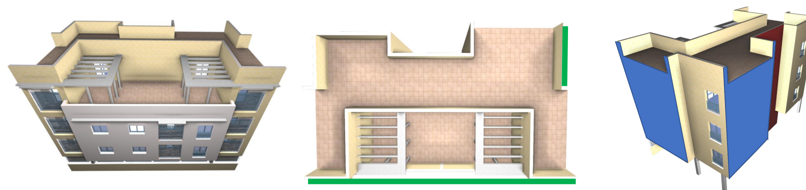
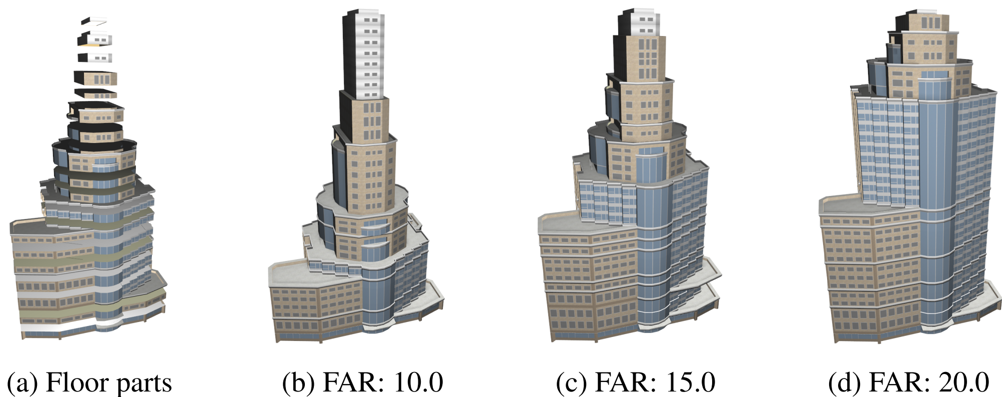


Figure 6.4: Building parts labeled as t-parts. This building parts have three neighboring elements and thus three cross-sections. This special building part has two length attributes as highlighted by the light green bar.

## 6.3 Case Study: Resource Constraint Building Design

We use our technique in order to synthesize a portfolio of different building designs from individual libraries of building parts. We show how using our concept of *structure*, global and local design goals can be implemented elegantly. One such example can be seen in Fig. 6.5, where we synthesized different skyscrapers from a set of floor parts shown in Figure 6.5 (a). Each of these floor parts is annotated with building height and a floor-area attribute round to a multiple of $0.2m$ and $1m^2$ respectively. In order to compute these building we have used two constraints *building height* and *FAR*. The resulting skyscrapers in Fig. 6.5 (b), (c) and (d) where forced to have the same height but the floor-to-area ratio (FAR, the total floor area divided by ground floor area) is varied. As one would expect, choosing a small FAR results in a more slender building, while the larger the FAR is chosen, the more chubby the shape of the building becomes.



(a) Floor parts (b) FAR: 10.0 (c) FAR: 15.0 (d) FAR: 20.0

Figure 6.5: Skyscrapers with a fixed height of 120 m and different floor-to-area ratios (FAR). Database: 18 elements, 80 transitions.

A second use case for architectural geometry is the construction of more complex building layouts from a user-defined footprint shape. We represent this footprint shape as a simple skeleton graph that describes the shape of the output building (see Figure 6.6, 6.8a, 6.7). The result shown in Figure 6.6 and 6.8a only use length as single global constraint. The result in Figure 6.7 uses length as a global constraint and local constraints by requesting specific attributes that are present within the different building parts. Vertices in the skeleton represent key points, and edges connecting them can be annotated with multiple additional constraints that need to be fulfilled, as well as an orthogonal direction vector to define the building front.

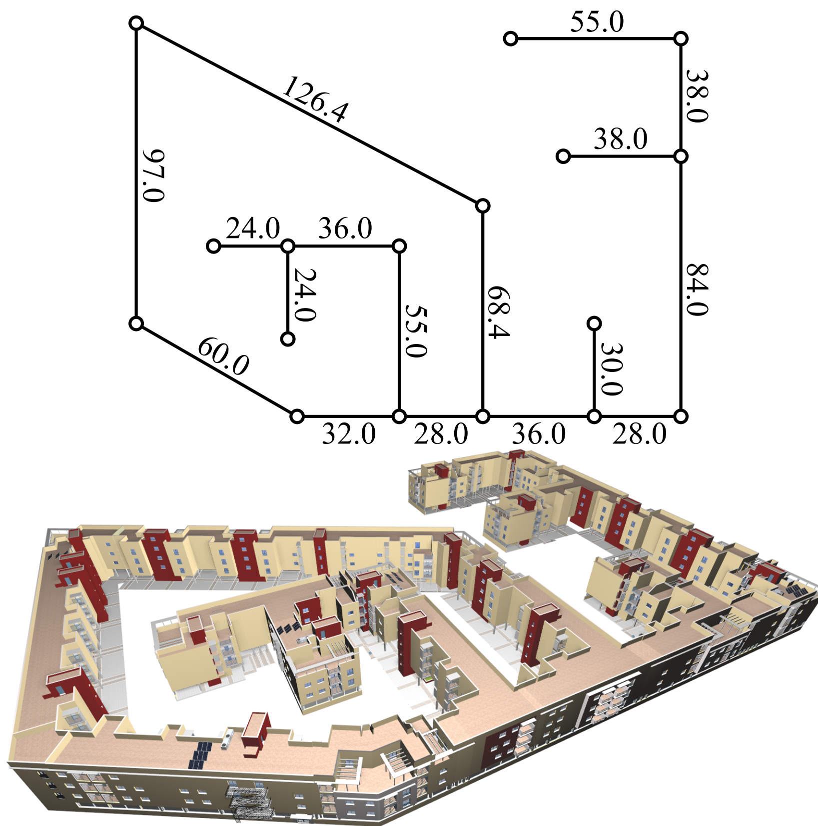



Figure 6.6: A complex example that contains multiple nested t-parts in combination with a closed loop.

At each key point, only a specific class of building parts is allowed to be positioned, which we determine by analyzing the number of outgoing edges of the skeleton vertices. If one edge leaves, either left or right end parts (see Figure 6.1) will be chosen, depending on the front direction of the edge. If two edges leave a vertex, *corner parts* (see Figure 6.2) depending on the angle between these edges are selected. *T-shaped* parts ((see Figure 6.4)) are selected from the database for vertices with three outgoing edges. From that input we compute a hierarchical structure according to Section 3.3.4 serving as input for the graph construction (see Section 3.3.1).

From the resulting intermediate graph (see section 3.3.1), the $k$-shortest-paths algorithm of Eppstein [Epp94] computes either the optimal sequence $X_{\text{opt}}$ or the $k$ best sequences that contain valid arrangements of building parts according to the input skeleton and the user-defined global and local constraints. As the solution found by the shortest path algorithm is a sequential list of building parts, we need to arrange them according to the input skeleton as a final step. Each key point el-

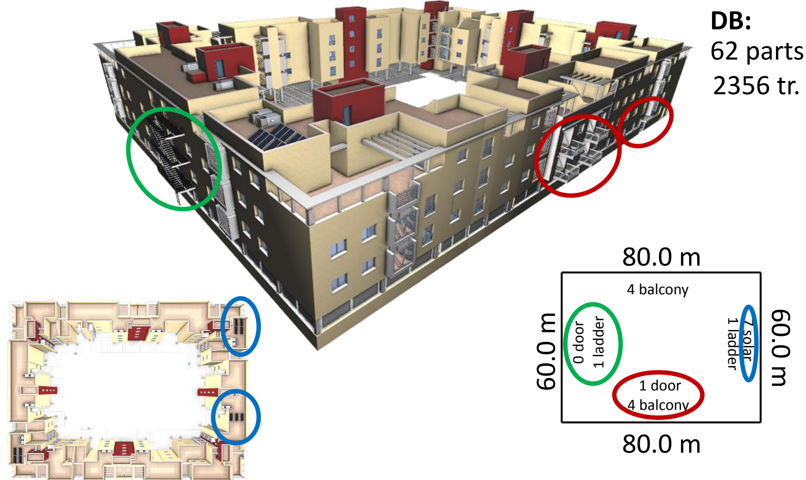



Figure 6.7: A building synthesized with different local constraints.

ement in the solution is aware at which vertex of the input skeleton it needs to be positioned, and so we can compute partial buildings starting and ending with key point elements, and simply transform them to the corresponding skeleton edges.

Decoupling the database and the defined structure makes it easily possible to synthesize buildings of the same shape targeting a completely different style, as illustrated in Fig. 6.8. Note: Both buildings have exactly the same shape, but the number of elements chosen to realize the shape is completely different (colored cylinder represent the start of a new element in the resulting sequence). This is why structure needs to be defined on the primary dimension rather than the sequence index (section 3.3.4). The database used to synthesize these buildings were composed of 99 elements, 4950 transitions in case of the yellow building and 131 elements, 9170 transitions in the case of the ancient building.

Variations can be achieved either by modifying the constraints as already demonstrated with the skyscraper model or by computing the $k$-best solutions and selecting according to taste. Fig. 6.9 demonstrates the three best solutions that were computed according to the defined cross-section disagreement cost function. All three buildings share exactly the same shape. By inspecting different limbs of the model and the parts selected at the t-junction and their surroundings, one can easily recognize that all three models vary their look locally. Such automatically generated variations may also serve as an inspiration to impose certain constraints on the next design iteration. For instance, the user may have noticed a balcony, but prefer to have it in a different location.

Finally, we note that despite the simple nature of our user input (key points and constraints), this concept of structure allows for intuitive modeling even of very

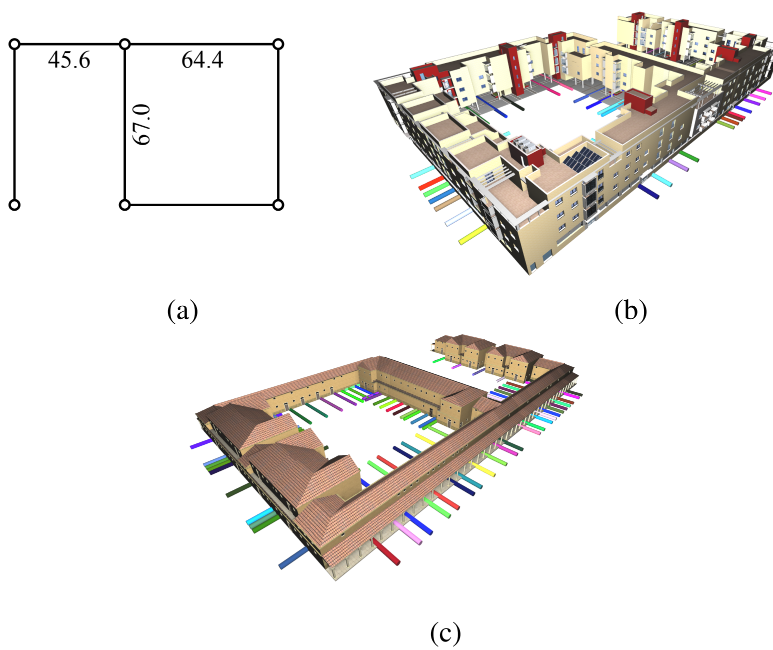



Figure 6.8: Buildings obtained from the same structure using different styles. The colored marks indicate that the spatial composition of each sequence is highly dependent on the part database.

complex buildings (Fig. 6.6, 6.5, 6.8 and 6.9). All models shown took no more than a few seconds to generate on a standard desktop PC (less than 3 s for the most complex example in Fig. 6.6), which makes our technique a candidate for interactive modeling sessions. In Fig. 6.12, we provide a rudimentary performance analysis across an extended range of problem sizes.

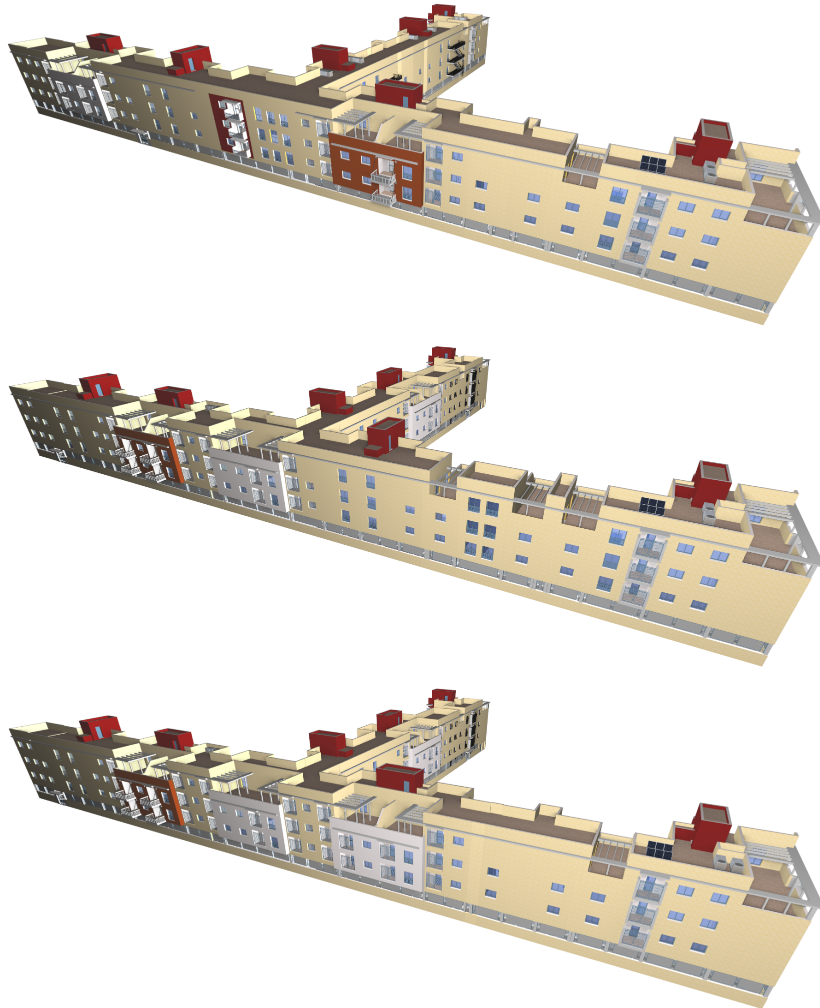


Figure 6.9: Three of the five best solutions given the same structure. Note that even the part at the T-joint can vary (also see Section 3.3.4).

It might happen that city blocks within an existing urban layout do not contain any building footprint at all (see Figure. 6.10). In order to fill such empty city blocks, we can use the building design technique presented in this chapter to synthesize a specific building. The design specification can be generated from the initial city block shape, i.e., the green polygon illustrated in Figure 6.10, described by the

street center lines.

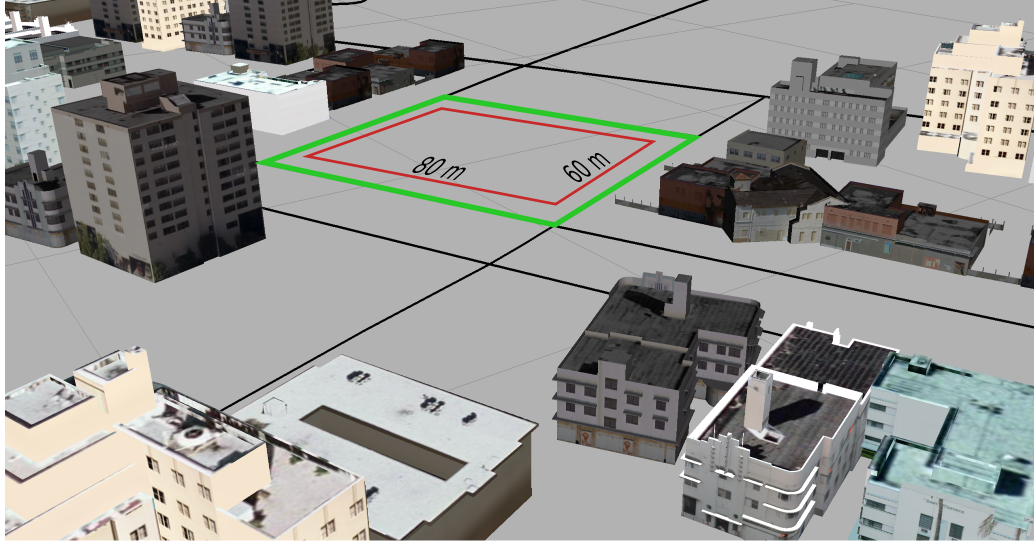



Figure 6.10: For empty city blocks such as the one highlighted by the green polygon, a building design specification can be automatically generated by computing an offset polygon (red)

The skeleton of the target building is defined by an offset polygon (see red polygon in Figure 6.10) computed from the initial city block shape. The building that was generated using the offset polygon as the design specification is illustrated in Figure 6.11. The yellow building is integrated into a city block layout that was populated with 3D buildings using the technique presented in section 5.1.

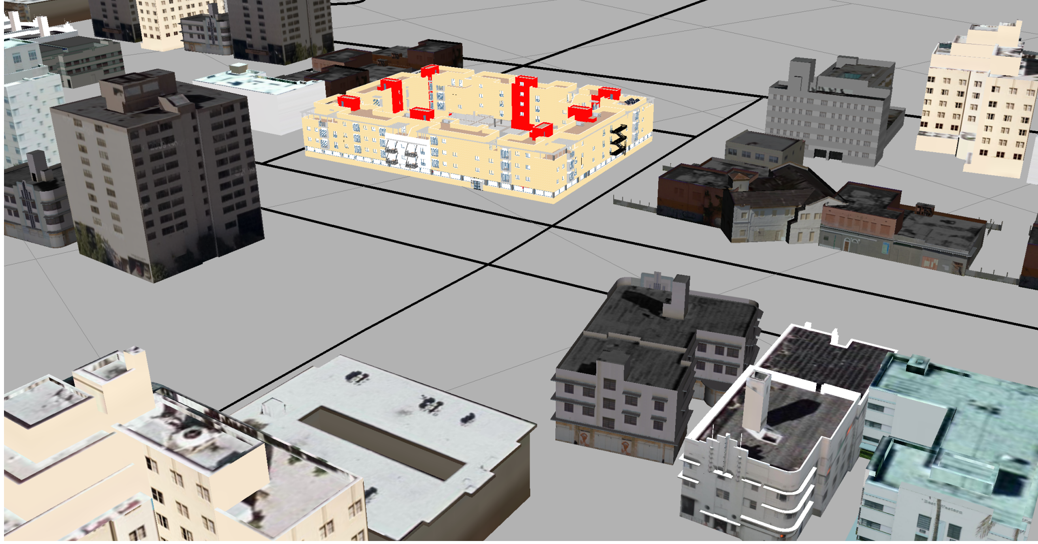
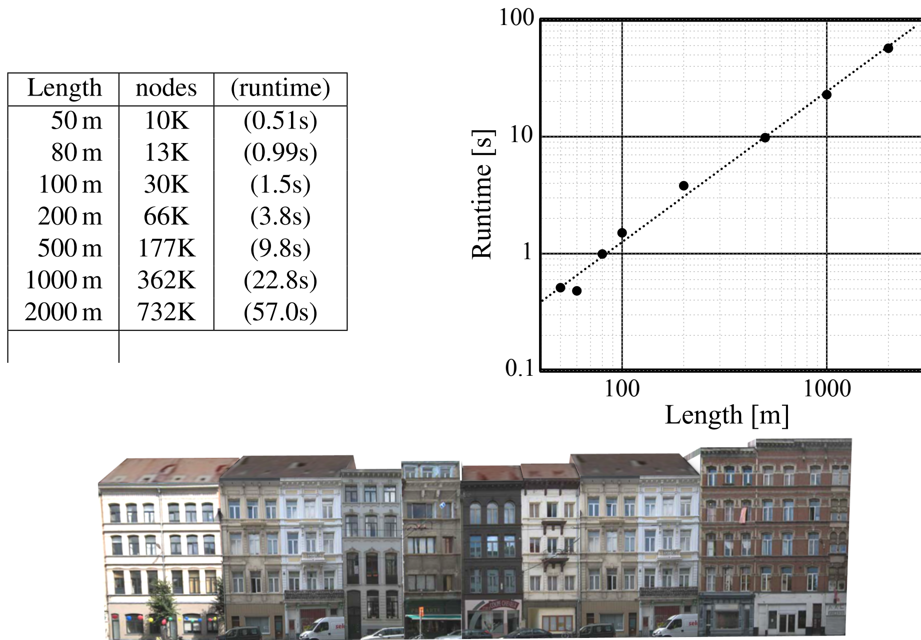


Figure 6.11: The synthesized yellow building integrated into an existing city block layout that was populated with 3D building models using the technique described in section 5.1

## 6.4 Performance Evaluation

To demonstrate the effectiveness of our approach we performed two series of experiments. First, we compared our method to related approaches in simple settings, where the variations of the path ranking method used in previous work can also find solutions. It should be noted that in the very rare case, when most of the elements concatenate with zero cost, path ranking method often finds the optimal solution faster than our algorithm, because in such case the probability is very high to find a feasible solution in one of the front places of the queue, used to enumerate sequences. Second, we generated various examples on more complex settings, for which path ranking fails to find a solution in a reasonable amount of time.



| Length | nodes | (runtime) |
|---|---|---|
| 50 m | 10K | (0.51s) |
| 80 m | 13K | (0.99s) |
| 100 m | 30K | (1.5s) |
| 200 m | 66K | (3.8s) |
| 500 m | 177K | (9.8s) |
| 1000 m | 362K | (22.8s) |
| 2000 m | 732K | (57.0s) |



Figure 6.12: Performance study: From a database of complete buildings (74 elements, 5402 transitions), we generated streets of varying length $R$ and measured the total time required for building the intermediate graph and finding the shortest path. The image shows the solution for 80 m. We found the runtime to be roughly proportional to $R^{1.29}$.

## 6.5 Analysis and Comparison

From literature, we are aware that buildings and/or facades can be modeled using forward [MWH+06, MM09] or inverse [BWS10, TLL+11] procedural modeling metaphors. Although they might be able to generate visually compelling results, we argue that these approaches are not directly suitable for modeling a combination of global and local constraints such as the floor-to-area ratio or the specific occurrence of architectural elements as shown in our examples (see Fig. 6.7). Another possibility would be utilizing stochastic tiling as proposed by Yeh et al. [YBY+13], which might indeed generate plausible results; however, their method would struggle to satisfy hard constraints.

## 6.6 Discussion and Future Work

**Availability of annotated data:** The outcome of any example based synthesis technique can only be as good as the data fed into it. In our case, the elements need to be meaningfully segmented and annotated with additional information, which is not included in the research databases we are aware of. Consequently, we had to manually prepare the 3D and motion data in order to make it usable for our purpose. The automatic and community-driven generation of example databases are vibrant research topics, and we are confident that in near future good example data will be available.

**Constraining the search space:** It is the nature of hard constraints that they cut down the space of feasible solutions. This, of course, depends on many factors including the size of the database and the relative resource consumption of the elements contained in it. In fact, in our experiments, we encountered cases where a very slight variation of equality constraints made all the difference between there being many solutions and none at all. Our algorithm performs best at equality constraints where suitable solutions exist. If there are none, for instance when trying to build a 50 m street out of a database of 20 m buildings, the failure can easily be detected by observing that the forward and backward trees do not connect. At that point, the user can be presented with several options to re-specify the problem, for instance by relaxing the constraint that caused the impossibility.

**Characterization:** Although we have presented a theoretical basis for handling more general structures including T-junctions under multiple constraints, our analysis of the algorithmic complexity is currently limited to the single constraint. From our experiments, we observe that the method scales well to rather complex structures including multiple T-joints. A rigorous analysis of the algorithmic and memory complexity in these scenarios is subject of future work.

**Transfer to applications:** The problem (RCSP) that our algorithm solves at its core is not specific to an application. However, due to the discrete way in which we handle resource usage, some creative experimentation may be needed in order to adapt the technique for a given content generation task. As more problems are solved using our technique, we hope to gain the insight required to make recommendations on how to best deal with certain types of constraints. So far, we found that most are best formulated in terms of per-element resources, and others may map better to per-transition costs. Yet others may depend on the context and require a side tap into the algorithm, like the extra cost term we used to make the animation follow a path—an example we used to illustrate that the core of our technique is in principle flexible enough to allow for the injection of other kinds of constraints.

**Outlook:** So far, we have only started exploring the potential of our algorithm in real-world use cases, and there are many research directions that might be worth a closer look. For instance, we are not aware of any example-based synthesis techniques for multi-character animations (imagine a dancing couple), which could

map well to motion graphs with T-joints as the actors split and re-join. Beyond the directions sketched in this paper, we plan to investigate the wider field of media computing. We see plenty of sequential problems that might benefit from our approach, from re-mixing of text, summarization of audio and video, to the generation of playlists or game level design [HF12a, STWM09].

# CHAPTER 7

# Summary

## 7.1 Conclusion

In here, we draw a detailed conclusion about the urban structure synthesizers presented in, chapter 3, chapter 4, chapter 5, and chapter 6.

**Example-based Road Network Synthesis** The proposed hierarchical street network generation (see section 4.1) is an example-based road network generation algorithm that only requires a limited set of user input, i.e., an arterial street skeleton and desired a topography map. These two types of user input are already enough to compute a detailed street network that faithfully reproduces the style found in the exemplars. We showed that the algorithm is able to synthesize road networks that resemble the style of a real-world network by using a novel hierarchical fragment representation capturing the road network styles at different scales. Our method relies on rich variations of fragments extracted from real-world cities to produce plausible road networks. Too few fragments in the database will typically result in large shape difference between the query and the matched fragments. As a consequence, the street segments will excessively be deformed, especially at the fragment boundaries. Although structures such as plazas, ramps, and roundabouts are topologically preserved they receive geometric distortions, that might produce unpleasant artifacts. However, the work of Yang et al. [YWVW13] showed that such warping effects can be drastically reduced when an iterative warping approach is used. In contrast, our algorithm can only compensate such warping artifacts by additional fragments. These need to be extracted from multiple cities with a similar street network style to have access to an increased amount of shape variations and thus more potential candidates. Optionally, a supervised fragment augmentation by scaling or mirroring might also help to enrich the variations present within

the database further.
The road network generation algorithm using recently introduced generative adversarial networks (GANS) (see section 4.2) is one of the very first attempts to use neural networks for the automatic content generation in the context of urban modeling. Standard deep learning techniques typically rely on images as input. Thus they cannot directly be used with road network representations available from *OpenStreetMap* or *EarthExplorer*. A transformation of the road network into the image domain using rasterization helped to overcome this problem. In contrast to other image synthesis techniques that train GANs on colored textures [JBU16], only the road network information is present within the input images. Thus, the network only 'sees' the road structure during the training and thus is able to adapt the filter banks to capture the details found in the road structure fully. After the training, only the generator network is kept and feeding it with samples from a simple distribution allowed to synthesize images containing road networks within milliseconds. To perform an evaluation of how well the statistics and network measures on the generated road network patches were reproduced we realized an effective post-processing algorithm to extract a graph-based representation of the synthesized network. The synthesized results that are produced by the generator are structurally sound and visually similar when compared to the example network. An in-depth statistical evaluation of different road network measures such as *city block area*, *city block compactness*, and *city block aspect ratio* was presented. The performed evaluation substantiated that the produced road networks are visually plausible, but also provided similar statistics as they can be found within the original road network. One advantage of the GAN technique used to realize the example-based road network synthesizer is indeed the capability to produce arbitrarily sized output patches. This allowed generating samples that are significantly larger than the input example. However, this also allowed generating patches, that are much smaller than the road network used for training. In both cases, these patches still contain meaningful road structures and might be suitable to be used in a synthesis application where the road network is grown from one or multiple seed patches. The lack of control over the generated content output was identified as a major drawback of this road network generation approach.

**Example-based City Block Layout Transfer and Re-synthesis** The proposed city block layout transfer algorithm (see 5.1) is an attempt to enrich city layouts with building footprint arrangements of those commonly found in real-world cities. The key idea of the proposed approach is to extract city blocks and their corresponding building footprint arrangements and re-use them to populate the empty city blocks within a synthesized road network with realistic building footprint distributions. The envisioned task was achieved by comparing the shape between

virtual and example city blocks, align them, and copy the content to the virtual city block. Conflicted buildings, i.e., building footprints that are partially located outside the virtual city block were removed, while building footprints that partially overlap the street area are corrected by a constraint optimization scheme based on position dynamics. Additional geometric details were added to the layout, by populating the building footprints with 3D buildings retrieved from a database. Here, we pursued the same retrieval strategy based on comparing the shape of the footprints to retrieve viable candidate buildings, as it was done during the city block layout transfer. In both cases: during the retrieval of city blocks and during the retrieval of candidate buildings our algorithm strongly depends on a rich variation of city block shapes as well as example building footprint shapes. Too few candidate shapes stored within the database will typically result in a large shape difference between the query object and the best-matched candidate object, city blocks and buildings respectively. In order to alleviate these problems, fall-back algorithms for city block layout re-synthesis (see section 5.2) and constrained building synthesis (see section 3 and 6) are proposed in the work at hand.

A re-synthesis scheme to compute novel city block layouts based on existing ones was presented and evaluated in section 5.2. With access to plenty of existing city block layouts from mapping services such as *OpenStreetMap*, the primary motivation for this approach was to re-use these layouts and transfer them to city blocks that strongly vary in shape. One of the key goals was to preserve the shape of the building footprints inside the new layout. Thus, using a warping algorithm to compute the novel layout was not an option. However, mapping the building footprints from the example city block into the target domain by applying a deformation, led to the introduced concept of the guidance map. This information has proven to be a powerful prior for the layout computation. At the very first step, a city block was used, and no deformation was applied. This step allowed visualizing and inspecting the differences between the original building footprint layout the synthesized one. In the result section, we demonstrated that the example-based modeling methodology can be used to compute new layouts in city blocks with a strong shape difference. The results presented in section 5.2 illustrate that the style of the layout present inside the original city block layout can be preserved even under complex deformations. The synthesis technique allows for exploring the layout space of different deformations at interactive rates, while whole city blocks can be synthesized within milliseconds. These fast generation times allow to integrate the algorithm into the urban layout generator presented in chapter 4.1. The most prominent drawback is that the algorithm relies on the presence of building footprints positioned along the road segments. Thus, utterly unstructured city blocks or city blocks with only large and irregular building footprints cannot be re-synthesized.

**Constrained Building Layouts** A novel algorithm for the constrained building generation was proposed in chapter 3 and chapter 6. The algorithm expects as input a database of annotated building parts. A specification of the abstract building shape in combination with the definition of the constraints, the algorithm first spans the space of resource feasible solutions as a graph. After this step, efficient path-search algorithms can be used to extract one or *k*-optimal solutions. First spanning the space of feasible solutions in a first step and then computing the best-solutions showed superior performance compared to path-ranking and forward path-search (see section 6.4). We introduced tree-decomposition and loop-cuts to synthesize 1.5D structures such as t-shaped buildings and even buildings with backyards or combinations of both. Exchanging the database and using the same specification enables to transfer the style of one building to a new shape. The abstract representation of the buildings in the form of annotations even allows tackling different types of content as it was exemplified in the corresponding paper [HTK+15] by synthesizing constraint motions along different curves. As a drawback, we identified that in cases, where no combination of parts will satisfy the constraints, we will not be able to produce a solution, because the forward and the backward search trees do not establish a connection. Here, a constraint relaxation according to user-defined importance might be a viable solution that needs to be evaluated in future work.

## 7.2 Example-Based Urban Modeling

The algorithms presented in this thesis are designed that they can be combined into an interactive modeling system for the generation of virtual urban environments. This allows providing an example-based modeling pipeline with a very flat learning curve and a slim user interface. In addition, it enables non-expert users to design virtual urban environments efficiently. The individual components for road network synthesis, city block layout synthesis, and the constrained building synthesis, only rely on a small set of parameters that might be hidden from a novice user. The components are independent of each other; therefore, each component might be exchanged by a different or improved version of the algorithm for a specific content type.

In recent years the user count for online mapping services such as *OpenStreetMap* has tremendously increased when reviewing their mapping statistics [Ope17b]. Thus, plenty of data for virtual urban structures is available to feed the road network synthesis algorithms (see chapter 4) and the city block re-synthesis scheme (see chapter 5). While the road network data from *OpenStreetMap* is available for a huge number of cities all over the world, the amount of building footprints and parcel information is still far from being complete. Especially, when building

attributes should be used one can notice that these annotations are incoherent and only sparsely available. For these reasons, the city block re-synthesis scheme unrolls its full potential, when these data might be available in the near future. The building models from *Trimble Warehouse* [Tri17] that were used to conduct the evaluation of the algorithm for the constrained building synthesis are handpicked examples that have high quality. The overall quality of the buildings available at *Trimble Warehouse* varies extremely, from very detailed models to buildings represented as textured boxes. All building parts that were used as database were manually prepared; however, for a practical use-case scenario, this manual preparation is not cost-effective. A digital content creation company, where such an algorithm might be used, typically has access to an internal database of building blocks that is already richly annotated. Even the building block have discrete dimensions to exchange and recombine them efficiently. These facts were confirmed in a private communication [cG17] with an industrial designer from a digital content creation company.

## 7.3 Technical Future Directions

There are several future directions that might be worth to be taken into account. In here, several future research directions for the methods proposed in the chapters 3, 4, 5, and 6 will be discussed. Finally, a bigger picture for research trends in the context of urban modeling will be sketched at the end of this chapter.

### 7.3.1 Road Network Synthesizers

The hierarchical road network generation (see section 4.1) approach might be extended by taking several future directions into account. To synthesize large-scale structures, i.e., streets that cross several districts and inter-fragment relationships, are crucial. Technically, this could be achieved by dropping the independent fragment insertion and replacing it by an algorithm that incorporates a cost term that measures potential street connections across the fragment boundaries. The algorithm could start with a randomly selected or user-specified region and grows the layout by simulating the insertion process while pairwise neighborhood costs are used to determine the best candidates. The algorithm might proceed greedily to produce a plausible street layout for each level. An alternative way would be to formulate the per-street level layout problem as a binary integer program, that incorporates a matching cost for each candidate fragment and the cost between the fragment boundaries. As off-the-shelve solvers have shown excellent performance for tiling problems [PYW14], network computation [PYB+16], and building reconstruction [NW17], the solver from Gurobi Inc. [Inc17] might be worth a closer

look for the evaluation of such an approach.
Apart from the synthesis, we would like to improve the usability of our approach by allowing the user to manipulate a synthesized layout. One method would be to enable the user to access multi-resolution editing tools for modifying road courses on different hierarchy levels efficiently. Typically, editing the road courses might require, local re-synthesis of affected regions, while incorporating both inter and intra-fragment context of the existing layout.
The road network generation algorithm proposed in section 4.2 might be improved by taking several directions into account. Real-world road networks are composed of streets having different importance categories, i.e., highways, connectors, or local streets. One interesting research direction would be to investigate, whether the GAN is able to learn road courses of multiple importance categories encoded into the same image. An important aspect that was not incorporated in this work is the controlled generation of a new road network. A user might sketch a road course that needs to be present in the final result, and the neural network should generate road courses that branch from the sketch to form a plausible extension of the road network. In combination with attribute maps [KAEE16], that might contain density information, land use, or even terrain more advanced control techniques would be possible. The GAN technique used in that part of the thesis allows generating output that has a smaller spatial resolution than the patches used for training. This enables the generation of individual road patches that might be used in an algorithm that grows a new road network similar to the approach of Nishida et al. [NGDA16a]. However, with the advantage that the templates contain a near infinite amount of variation assumed that the data-generating distribution has been successfully learned by the GAN. Finally, the rasterized road network needs a smart post-processing step to allow a fair comparison to existing road network generators. Apart from using GANs for urban structures, we might also investigate their use for feature map generation for texture synthesis algorithms such as [RSK10, KNL$^+$15].

### 7.3.2 Cityblock Layouts Synthesizers

The city block layout transfer algorithm (see section 5.1) might be improved in several ways. To improve the overall building layout within an urban region a possible step would be to integrate a statistical model for the distribution of buildings. This model might also take into account higher order structures of the underlying road network. It might be possible to learn such information from the example cities, by taking building density, building height, or land use into account. The prediction of the building locations might be realized by the development of a novel algorithm inspired by the recent work of Arietta et al. [AERA14]. In combination with the strategic placement of individual entities such as parks or

recreational areas, the overall realism of the urban layout can be drastically improved. Another direction would be the design of a descriptor for the comparison of the building footprint arrangement within a city block. This would then allow to constraining the selection of a viable city block to the adjacent city blocks, as in real-world city layouts the style in adjacent city blocks is typically coherent. An optimal selection of the best-suited city blocks according to shape and layout similarity might be achieved by formulating the city block layout problem might an integer linear program, that can be efficiently solved using off-the-shelve solvers such as the one from Gurobi Inc. [Inc17].
The layout computation for city blocks (see section 5.2) might be extended and improved in several ways. The proposed algorithm relies on building footprints that are arranged in sequences located near the boundary street segments. Thus, interior buildings are neglected or might only be copied to the target block. An essential improvement would be, to exchange the labeling algorithm with a more powerful algorithm that is able to synthesize more general two-dimensional layouts. For room layouts [YYT+11, MSL+11] or shop layouts [YYW+12] Markov Chain Monte Carlo (MCMC) algorithms have been explored. However, instead of steering these algorithms by rules or design guidelines the proposed guidance map might be a suitable alternative. Transferring the layout of a source city block to a target city block might also be achieved using learned position or orientation functions as proposed in Guerrero et al. [GJWW14, GJWW15]. In addition, recent machine learning techniques, such as GANs in combination with a large database of city blocks might be used to learn a layout model and to predict an occupancy map for different shaped city blocks. As individual city blocks are part of a larger neighborhood, contextual information from nearby locations might serve as additional information during the training procedure.

### 7.3.3 Constrained Building Synthesizers

There are several directions that would improve the constrained building synthesis approach (see chapter 3 and 6). If a user defines local and global constraints that cannot be satisfied, because there is no valid combination of building parts within the database, the algorithm is not able to produce a resource feasible solution. In such a case the forward and the backward search tree have not established a connection during the construction of the intermediate graph. A possible solution to tackle this problem would be a relaxation of the defined resource constraints, by letting a user specify different importance weights for the satisfaction of the different resource constraints. In a post-processing step, the intermediate graph could be analyzed to detect disconnections. A connection could be forced by adding placeholder nodes carrying virtual resources to fill the gap taking the importance weights into account. Another possibility to produce a resource feasible solu-

tion might be a relaxation of the globally optimal result. Instead of constructing the full intermediate graph, the substructures might be solved incrementally, and connected substructures are forced to start or end with the building parts of an adjacent solution. A similar approach has been successfully employed to synthesize game level layouts [MVLS14].
Apart from the building synthesis, the graph-based optimization technique (see chapter 3) might be utilized for other content domains, such as video summarization of multiple video streams. An application scenario would be fusing various captured video streams into a single video, while different time constraints for individual example streams might be specified and a global time limit should not be exceeded. Other application domains might be the constrained generation of song lists that consist of an overall duration time with constraints on songs from different music genres, or sequential game level layouts as they are common in 2d platform video games. For the latter domain, different approaches have already been proposed [HF12b, STWM09], that can be used as a starting point for an in-depth evaluation.

## 7.4 Future Trends for Example-Based Modeling

In the final section of these work, an outlook for future research directions on efficient digital content generation will be given.

### 7.4.1 Generation of Urban Structures

Research on the synthesis of road networks has spawned several approaches from the generation of cross-country roads to the generation of the individual street layouts of settlements or cities. An essential aspect of large-scale road networks, however, is the connection of cross-country roads with the outgoing links of an existing city. Algorithms that can establish such nontrivial connections need to provide generic parametric models for the generation of motorway interchanges and on-ramps. These parametric structures might be used by algorithms that are designed to compute a strategic placement of these structures. The placement strategy might include contextual information of the existing city layout such as the presence and the accessibility of industrial areas, recreational areas, and park and rail access points. In urban environments, multiple modalities for transportation exist aside from road networks. Examples for these modalities are railways, underground railways, trams, cycleways, and bus networks. In large-scale urban environments, these structures partially depend on each other, to ensure time-efficient movement. However, the possible placement of these structures strongly depends on the environment and might increase their construction costs. For the

planning of multi-mode transportation networks, it is crucial to design novel synthesis algorithms that might incorporate such dependencies. In addition, multi-model synthesis algorithms might also need different user interaction methodologies to enable the possibility to specify the functional demands and the constraints to control the complex content.

### 7.4.2 Content Storage and Compression

Apart from the design of novel content generation algorithms, it is essential to consider the storage size of the generated content. While the cost for storage space on hard-disks might be neglected, in-memory storage might still be limited to today's consumer hardware. With the increasing demand for additional details and the growing size of virtual worlds, future content generation algorithms will need to synthesize novel content in a compressed representation that is only unpacked during the visualization stage. When the content needs to be edited at runtime, algorithms need to efficiently update the compressed representation without regenerating the whole content and re-compression the generated data. For real-time texture synthesis, the first steps have already been taken into account in the work of Lefevbre et al. [LHL10]. For the large-scale synthesis of virtual worlds these crucial steps, have been neglected so far.

### 7.4.3 Level of Detail

Increased level of detail is essential to increase the perceived coherence within a virtual environment. However, manual placement of these additional details is too costly, especially when the size of the virtual world is huge. Algorithms for synthesizing specific details found in different contexts have already been presented. They range from synthesizing detailed geometry onto abstract shapes [MWT11], [ROM+15] using ideas from texture synthesis or point processes, or placing traffic signs along road networks [TB16], and the synthesis of entangled details for natural environments. Apart, from these details that already increase the realism of virtual urban environments, there are a few other types of details that make them perceived as real: imperfections, forfeit buildings and gardens, or even small piles of dirt and waste. Instead of adding these details afterward into a scene, content generation algorithms should be adapted that all these unpleasant details are present in the future virtual urban environments.

### 7.4.4 Neurally Guided Content Generators

Many of today's example-based content generation algorithms rely on a custom-tailored algorithm that is able to synthesize a specific type of content from pro-

vided examples. With the advent of deep learning and especially the invention of generative adversarial networks (GAN) by Goodfellow [GPAM$^{+}$14] there is a huge amount of unexplored potential for urban content generation and virtual asset generation algorithms in general. The road network synthesis algorithm based on GANs is an early step towards the replacement of the custom-tailored example-based content generators by learned models. Typically, content that can be transformed into an image or a volume representation might be suited for training such a generative model. Recent works in this field range from terrain synthesis [BP17] to 3d shape synthesis [WZX$^{+}$16, LXC$^{+}$17].
Classical neural networks, for example, have been applied to the generation of resource maps for video games [LIHT16]. Such an approach could in principle be used to learn the strategic placement of individual urban objects, such as parks, recreational areas, or schools.
Although the number of papers that use generative adversarial networks in particular for the generation of content has drastically increased recently [Kal17], there are still plenty of unexplored areas where the use of a GAN might be useful, such as 2D and 3D layouts, distributions of plants or trees. However, besides finding improved training strategies, sophisticated network architectures, one focus of attention needs to be on the design of smart post-processing algorithms that improve the output quality of generative component to make the produced content directly usable in specific applications.

# Bibliography

[AAAG96] Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. A novel type of skeleton for polygons. In *J. UCS The Journal of Universal Computer Science*, pages 752–761. Springer, 1996.

[ADBW16] Daniel G Aliaga, İlke Demir, Bedrich Benes, and Michael Wand. Inverse procedural modeling of 3d models for virtual worlds. In *ACM SIGGRAPH 2016 Courses*, page 16. ACM, 2016.

[AERA14] Sean M Arietta, Alexei A Efros, Ravi Ramamoorthi, and Maneesh Agrawala. City forensics: Using visual elements to predict non-visual city attributes. *IEEE transactions on visualization and computer graphics*, 20(12):2624–2633, 2014.

[AVB08] Daniel G. Aliaga, Carlos A. Vanegas, and Bedrich Benes. Interactive example-based urban layout synthesis. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, pages 160:1–160:10, New York, NY, USA, 2008. ACM.

[BA05] Daniel R. Bekins and Daniel G. Aliaga. Build-by-number: Rearranging the real world to visualize novel architectural spaces. In *IEEE Visualization*, page 19. IEEE Computer Society, 2005.

[BBL+17] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[BLS16] Filip Biljecki, Hugo Ledoux, and Jantien Stoter. Generation of multi-lod 3d city models in citygml with the procedural modelling engine random3dcity. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 3(4), 2016.

[BMM00] Serge Belongie, Greg Mori, and Jitendra Malik. Matching with shape contexts. In *Proceedings of the IEEE Workshop on Content-based Access of Image and Video Libraries*, 2000.

[BP17] Christopher Beckham and Christopher Pal. A step towards procedural terrain generation with gans, 2017.

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, November 2001.

[BWK14] Jan Bene, Alexander Wilkie, and Jaroslav Kivnek. Procedural modelling of urban road networks. *Computer Graphics Forum*, 33(6):132–142, 2014.

[BWS10] Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. A connection between partial symmetry and inverse procedural modeling. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 104:1–104:10, New York, NY, USA, 2010. ACM.

[CEW+08] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller, and Eugene Zhang. Interactive procedural street modeling. In *ACM transactions on graphics (TOG)*, volume 27, page 103. ACM, 2008.

[cG17] cantaloupe GmbH. Private communication. Website, 2017. available `https://www.cantaloupe.de`; accessed August, 5th, 2017.

[CMK+14] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[DAB15] Ilke Demir, Daniel G Aliaga, and Bedrich Benes. Procedural editing of 3d building point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2147–2155, 2015.

[DP73] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[DRSV13] D. Dai, H. Riemenschneider, G. Schmitt, and L. Van Gool. Example-based facade texture synthesis. In *International Conference on Computer Vision (ICCV)*, 2013.

[DRV14] D. Dai, H. Riemenschneider, and L. Van Gool. The synthesizability of texture examples. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[Ear17] EarthExplorer. Website, 2017. available `https://earthexplorer.usgs.gov/`; accessed July, 15th, 2017.

[EBP+12] Arnaud Emilien, Adrien Bernhardt, Adrien Peytavie, Marie-Paule Cani, and Eric Galin. Procedural generation of villages on arbitrary terrains. *The Visual Computer*, 28(6-8):809–818, 2012.

[Epp94] David Eppstein. Finding the *k* shortest paths. In *Proc. 35th Symp. Foundations of Computer Science*, pages 154–165. IEEE, November 1994.

[Esr17] Esri. City engine. Website, 2017. available `http://www.esri.com/software/cityengine`; accessed July, 15th, 2017.

[EVC+15] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. Worldbrush: Interactive example-based synthesis of procedural virtual worlds. *ACM Trans. Graph.*, 34(4):106:1–106:11, July 2015.

[FBG+16] Marek Fiser, Bedrich Benes, Jorge Garcia Galicia, Michel Abdul-Massih, Daniel G. Aliaga, and Vojtech Krs. Learning geometric graph grammars. In *Proceedings of the 32Nd Spring Conference on Computer Graphics*, SCCG '16, pages 7–15, New York, NY, USA, 2016. ACM.

[FKS+04] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 652–663. ACM, 2004.

[FRS+12] Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. Example-based synthesis of 3d object arrangements. *ACM Trans. Graph.*, 31(6):135:1–135:11, November 2012.

[Gar09] Renan Garcia. *Resource constrained shortest paths and extensions.* PhD thesis, Georgia Institute of Technology, 2009.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GEB15] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *NIPS*, pages 262–270, 2015.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GJWW14] Paul Guerrero, Stefan Jeschke, Michael Wimmer, and Peter Wonka. Edit propagation using geometric relationship functions. *ACM Trans. Graph.*, 33(2):15:1–15:15, April 2014.

[GJWW15] Paul Guerrero, Stefan Jeschke, Michael Wimmer, and Peter Wonka. Learning shape placements by example. *ACM Trans. Graph.*, 34(4):108:1–108:13, July 2015.

[Goo16] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[GPAM$^+$14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. pages 2672–2680. Curran Associates, Inc., 2014.

[GPGB11] Eric Galin, Adrien Peytavie, Eric Guérin, and Bedřich Beneš. Authoring hierarchical road networks. In *Computer Graphics Forum*, volume 30, pages 2021–2030. Wiley Online Library, 2011.

[GPMG10] Eric Galin, Adrien Peytavie, Nicolas Maréchal, and Eric Guérin. Procedural generation of roads. In *Computer Graphics Forum*, volume 29, pages 429–438. Wiley Online Library, 2010.

[HF06] Kai Hormann and Michael S. Floater. Mean value coordinates for arbitrary planar polygons. *ACM Trans. Graph.*, 25(4):1424–1441, October 2006.

[HF12a] Ian D Horswill and Leif Foged. Fast procedural level population with playability constraints. In *AIIDE*, 2012.

[HF12b] Ian D Horswill and Leif Foged. Fast procedural level population with playability constraints. In *AIIDE*, 2012.

[HKK15] Stefan Hartmann, Björn Krüger, and Reinhard Klein. Content-aware re-targeting of discrete element layouts. In *International Conference on Computer Graphics, Visualization and Computer Vision*, volume 23 of *WSCG proceedings*, pages 173–182, June 2015.

[HKYM16] Haibin Huang, Evangelos Kalogerakis, ME Yumer, and Radomir Mech. Shape synthesis from sketches via procedural models and convolutional networks. *IEEE Transactions on Visualization and Computer Graphics*, 2016.

[Hoc91] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. diploma thesis. Master's thesis, Technische Universität Münschen, 1991.

[Hol94] Matthew Holton. Strands, gravity and botanical tree imagery. *Computer Graphics Forum*, 13(1):57–67, 1994.

[HTK$^+$15] Stefan Hartmann, Elena Trunz, Björn Krüger, Reinhard Klein, and Matthias B. Hullin. Efficient multi-constrained optimization for example-based synthesis. *The Visual Computer / Proc. Computer Graphics International (CGI 2015)*, 31(6-8):893–904, June 2015.

[HWWK17] Stefan Hartmann, Michael Weinmann, Raoul Wessel, and Reinhard Klein. Streetgan: Towards road network synthesis with generative adversarial networks. In *International Conference on Computer Graphics, Visualization and Computer Vision*, June 2017.

[HZ80] Gabriel Y. Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.

[Inc17] Gurobi Inc. Linear programming solvers. Website, 2017. available `http://www.gurobi.com/`; accessed August, 20th, 2017.

[IZZE16] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.

[JBU16] Nikolay Jetchev and Roland Bergmann Urs, Vollgraf. Texture synthesis with spatial generative adversarial networks. pages 2672–2680. Curran Associates, Inc., 2016.

[KAEE16] Levent Karacan, Zeynep Akata, Aykut Erdem, and Erkut Erdem. Learning to generate images of outdoor scenes from attributes and semantic layouts. *arXiv preprint arXiv:1612.00215*, 2016.

[Kal17] Grigorios Kalliatakis. Delving-deep-into-gans. Website, 2017. available `https://github.com/GKalliatakis/Delving-deep-into-GANs`; accessed August, 6th, 2017.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KGP02] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 473–482, New York, NY, USA, 2002. ACM.

[KHWM17] Jonathan Klein, Stefan Hartmann, Michael Weinmann, and Dominik L. Michels. Multi-scale terrain texturing using generative adversarial networks. In *Image and Vision Computing New Zealand*, December 2017.

[KM07] George Kelly and Hugh McCabe. Citygen: An interactive system for procedural city generation. In *Fifth International Conference on Game Design and Technology*, pages 8–16, 2007.

[KMK12] Lars Krecklau, Christopher Manthei, and Leif Kobbelt. Procedural interpolation of historical city maps. In *Computer Graphics Forum*, volume 31, pages 691–700. Wiley Online Library, 2012.

[KNL+15] Alexandre Kaspar, Boris Neubert, Dani Lischinski, Mark Pauly, and Johannes Kopf. Self tuning texture optimization. In *Computer Graphics Forum*, volume 34, pages 349–359. Wiley Online Library, 2015.

[LCL07] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. In *Computer Graphics Forum*, volume 26, pages 655–664. Wiley Online Library, 2007.

[LCOZ+11] Jinjie Lin, Daniel Cohen-Or, Hao Zhang, Cheng Liang, Andrei Sharf, Oliver Deussen, and Baoquan Chen. Structure-preserving retargeting of irregular 3d architecture. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pages 183:1–183:10, New York, NY, USA, 2011. ACM.

[LHL10] Sylvain Lefebvre, Samuel Hornus, and Anass Lasram. By-example synthesis of architectural textures. *ACM Transactions on Graphics (TOG)*, 29(4):84, 2010.

[LIHT16] Scott Lee, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. Predicting resource locations in game maps using deep convolutional neural networks. In *The Twelfth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. AAAI*, 2016.

[LKF12] Yaron Lipman, Vladimir G. Kim, and Thomas A. Funkhouser. Simple formulas for quasiconformal plane deformations. *ACM Trans. Graph.*, 31(5):124:1–124:13, September 2012.

[LSWW11] Markus Lipp, Daniel Scherzer, Peter Wonka, and Michael Wimmer. Interactive modeling of city layouts using layers of procedural content. In *Computer Graphics Forum*, volume 30, pages 345–354. Wiley Online Library, 2011.

[LW16] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *European Conference on Computer Vision*, pages 702–716. Springer, 2016.

[LXC+17] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. *arXiv preprint arXiv:1705.02090*, 2017.

[LZ10] Wan-Yen Lo and Matthias Zwicker. Bidirectional search for interactive motion synthesis. *Computer Graphics Forum*, 29(2):563–573, 2010.

[M00] Cohen M. Everything by example. Keynote, 2000. Keynote talk at Chinagraphics.

[Man11] Alexander Mandt. Topographische synthese von straennetzwerken. Diploma Thesis, 2011.

[Mar04] Stephen Marshall. *Streets and Patterns*. Spon Press, 270 Madison Ave, New York, NY 10016, 2004.

[MDWK08] Sebastian Möser, Patrick Degener, Roland Wahl, and Reinhard Klein. Context aware terrain visualization for wayfinding and navigation. *Computer Graphics Forum*, 27(7):1853–1860, October 2008.

[Mer07] Paul Merrell. Example-based model synthesis. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 105–112, New York, NY, USA, 2007. ACM.

[MHHR07] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, April 2007.

[MIS17] TANVI MISRA. X-ray your city's street network, 2017.

[MM08] Paul Merrell and Dinesh Manocha. Continuous model synthesis. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, pages 158:1–158:7, New York, NY, USA, 2008. ACM.

[MM09] Paul Merrell and Dinesh Manocha. Constraint-based model synthesis. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, SPM '09, pages 101–111, New York, NY, USA, 2009. ACM.

[MO14] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[MR84] Michel Minoux and Celso Ribeiro. A transformation of hard (equality constrained) knapsack problems into constrained shortest path problems. *Oper. Res. Lett.*, 3(4):211–214, October 1984.

[MRC+07] M. Müller, T. Röder, M. Clausen, B. Eberhardt, Björn Krüger, and Andreas Weber. Documentation mocap database hdm05. Technical Report CG-2007-2, Universität Bonn, June 2007.

[MSL+11] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph.*, 30(4):87:1–87:10, July 2011.

[MVLS14] Chongyang Ma, Nicholas Vining, Sylvain Lefebvre, and Alla Sheffer. Game level layout from design specification. In *Eurographics 2014*, pages 95–104, 2014.

[MWA+13] Przemyslaw Musialski, Peter Wonka, Daniel G. Aliaga, Michael Wimmer, Luc van Gool, and Werner Purgathofer. A Survey of Urban Reconstruction. *Computer Graphics Forum*, 32(6):146–177, September 2013.

[MWH+06] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 614–623, New York, NY, USA, 2006. ACM.

[MWT11] Chongyang Ma, Li-Yi Wei, and Xin Tong. Discrete element textures. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH 2011, pages 62:1–62:10, New York, NY, USA, 2011. ACM.

[MZWVG07] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[NGDA16a] Gen Nishida, Ignacio Garcia-Dorado, and Daniel G Aliaga. Example-driven procedural urban roads. In *Computer Graphics Forum*, volume 35, pages 5–17. Wiley Online Library, 2016.

[NGDA+16b] Gen Nishida, Ignacio Garcia-Dorado, Daniel G Aliaga, Bedrich Benes, and Adrien Bousseau. Interactive sketching of urban procedural models. *ACM Transactions on Graphics (TOG)*, 35(4):130, 2016.

[NW17] Liangliang Nan and Peter Wonka. Polyfit: Polygonal surface reconstruction from point clouds. 2017.

[OG12] A. Cengiz Öztireli and Markus Gross. Analysis and synthesis of point distributions based on pair correlation. *ACM Trans. Graph.*, 31(6):170:1–170:10, November 2012.

[Ope17a] OpenStreetMap. Website, 2017. available `http://www.openstreetmap.org`; accessed July, 15th, 2017.

[Ope17b] Mapping Statistics OpenStreetMap. Website, 2017. available `http://wiki.openstreetmap.org/wiki/Stats`; accessed August, 5th, 2017.

[Ops17] Site Ops. Site engineering software. Website, 2017. available `http://www.siteops.com/products/site-layout/`; accessed July, 15th, 2017.

[PGB03] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 313–318, New York, NY, USA, 2003. ACM.

[PM01] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, 2001. ACM.

[PMG+05] Mark Pauly, Niloy J Mitra, Joachim Giesen, Markus H Gross, and Leonidas J Guibas. Example-based 3d scan completion. In *Symposium on Geometry Processing*, number EPFL-CONF-149337, pages 23–32, 2005.

[PYB+16] Chi-Han Peng, Yong-Liang Yang, Fan Bao, Daniel Fink, Dong-Ming Yan, Peter Wonka, and Niloy J. Mitra. Computational network design from functional specifications. *ACM Trans. Graph.*, 35(4):131:1–131:12, July 2016.

[PYW14] Chi-Han Peng, Yong-Liang Yang, and Peter Wonka. Computing layouts with deformable templates. *ACM Trans. Graph.*, 33(4):99:1–99:11, July 2014.

[RM85] Celso C Ribeiro and Michel Minoux. A heuristic approach to hard constrained shortest path problems. *Discrete Applied Mathematics*, 10(2):125–137, 1985.

[RMC15] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[ROM+15] Riccardo Roveri, A. Cengiz Öztireli, Sebastian Martin, Barbara Solenthaler, and Markus Gross. Example based repetitive structure synthesis. *Comput. Graph. Forum*, 34(5):39–52, August 2015.

[RSK10] Roland Ruiters, Ruwen Schnabel, and Reinhard Klein. Patch-based texture interpolation. *Computer Graphics Forum (Proc. of EGSR)*, 29(4):1421–1429, June 2010.

[RSK13] Roland Ruiters, Christopher Schwartz, and Reinhard Klein. Example-based interpolation and synthesis of bidirectional texture functions. *Computer Graphics Forum (Proceedings of the Eurographics 2013)*, 32(2):361–370, May 2013.

[RTHG16] Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah D Goodman. Neurally-guided procedural models: learning to guide procedural models with deep neural networks. *arXiv preprint arXiv: 1603.06143*, 2016.

[S+85] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.

[SC78] H Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. on Acoust., Speech, and Signal Process.*, 26:43–49, 1978.

[SG89] C. C. Skiscim and B. L. Golden. Solving k-shortest and constrained shortest path problems efficiently. *Ann. Oper. Res.*, 20(1-4):249–282, August 1989.

[SH07] Alla Safonova and Jessica K. Hodgins. Construction and optimal search of interpolated motion graphs. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[SKUF15] C Schinko, U Krispel, T Ullrich, and D Fellner. Built by algorithms-state of the art report on procedural modeling. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(5):469, 2015.

[SM15] Michael Schwarz and Pascal Müller. Advanced procedural modeling of architecture. *ACM Transactions on Graphics (TOG)*, 34(4):107, 2015.

[Sof17] Autodesk Software. Civil 3d. Keynote, 2017. available `https://www.autodesk.de/products/autocad-civil-3d/overview`; accessed July, 15th, 2017.

[SSD+17] Felipe Petroski Such, Shagan Sah, Miguel Dominguez, Suhas Pillai, Chao Zhang, Andrew Michael, Nathan Cahill, and Raymond Ptucha. Robust spatial filtering with graph convolutional neural networks. *arXiv preprint arXiv:1703.00792*, 2017.

[STWM09] Gillian Smith, Mike Treanor, Jim Whitehead, and Michael Mateas. Rhythm-based level generation for 2d platformers. In *Conf. on Foundations of Digital Games*, pages 175–182, 2009.

[SYBG02] Jing Sun, Xiaobo Yu, George Baciu, and Mark Green. Template-based generation of road networks for virtual city modeling. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '02, pages 33–40, New York, NY, USA, 2002. ACM.

[SYZ+17] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[TB16] Fieke Taal and Rafael Bidarra. Procedural generation of traffic signs. In *Eurographics Workshop on Urban Data Modelling and Visualisation*. Eurographics, Eurographics, dec 2016.

[TC89] C-H Teh and Roland T. Chin. On the detection of dominant points on digital curves. *IEEE Transactions on pattern analysis and machine intelligence*, 11(8):859–872, 1989.

[Tex17] Texture.com. Website, 2017. available `https://textures.com/`; accessed July, 15th, 2017.

[TLL+11] Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun. Metropolis procedural modeling. *ACM Trans. Graph.*, 30(2):11:1–11:14, April 2011.

[Tri17] Trimble. Warehouse3d. Website, 2017. available `https://3dwarehouse.sketchup.com/`; accessed July, 15th, 2017.

[Tur12] Lara Turner. Variants of shortest path problems. *Algorithmic Operations Research*, 6(2):91–104, 2012.

[Tur17] Turbosquid.com. Website, 2017. available `https://www.turbosquid.com`; accessed July, 16th, 2017.

[VABW09a] Carlos A Vanegas, Daniel G Aliaga, Bedrich Benes, and Paul Waddell. Visualization of simulated urban spaces: Inferring parameterized generation of streets, parcels, and aerial imagery. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):424–435, 2009.

[VABW09b] Carlos A Vanegas, Daniel G Aliaga, Bedrich Benes, and Paul A Waddell. Interactive design of urban spaces using geometrical and behavioral modeling. In *ACM Transactions on Graphics (TOG)*, volume 28, page 111. ACM, 2009.

[VAW+10] Carlos A Vanegas, Daniel G Aliaga, Peter Wonka, Pascal Müller, Paul Waddell, and Benjamin Watson. Modelling the appearance and behaviour of urban spaces. In *Computer Graphics Forum*, volume 29, pages 25–42. Wiley Online Library, 2010.

[Vit67] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[VKW+12] Carlos A Vanegas, Tom Kelly, Basil Weber, Jan Halatsch, Daniel G Aliaga, and Pascal Müller. Procedural generation of parcels in urban modeling. In *Computer graphics forum*, volume 31, pages 681–690. Wiley Online Library, 2012.

[Wad02] Paul Waddell. Urbansim: Modeling urban development for land use, transportation, and environmental planning. *Journal of the American planning association*, 68(3):297–314, 2002.

[WAMV11] Peter Wonka, Daniel Aliaga, Pascal Müller, and Carlos Vanegas. Modeling 3d urban spaces using procedural and simulation-based techniques. In *ACM SIGGRAPH 2011 Courses*, page 9. ACM, 2011.

[WBN+03] Paul Waddell, Alan Borning, Michael Noth, Nathan Freier, Michael Becke, and Gudmundur Ulfarsson. Microsimulation of urban development and location choices: Design and implementation of urbansim. *Networks and spatial economics*, 3(1):43–67, 2003.

[WBSH+13] Simon Wenner, Jean-Charles Bazin, Alexander Sorkine-Hornung, Changil Kim, and Markus Gross. Scalable music: Automatic music retargeting and synthesis. In *Computer Graphics Forum*, volume 32, pages 345–354. Wiley Online Library, 2013.

[WLKT09] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*, pages 93–117. Eurographics Association, 2009.

[WLW+14] Xiaokun Wu, Chuan Li, Michael Wand, Klaus Hildebrandt, Silke Jansen, and Hans-Peter Seidel. 3d model retargeting using offset statistics. In *3D Vision (3DV), 2014 2nd International Conference on*, volume 1, pages 353–360. IEEE, 2014.

[WMWF07] Peter Wonka, Pascal Müller, Ben Watson, and Andy Fuller. Urban design and procedural modeling. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pages 229–229, New York, NY, USA, 2007. ACM.

[WMWG09] Basil Weber, Pascal Müller, Peter Wonka, and Markus Gross. Interactive geometric simulation of 4d cities. In *Computer Graphics Forum*, volume 28, pages 481–492. Wiley Online Library, 2009.

[WWSR03] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant architecture. In *SIGGRAPH 2003*, SIGGRAPH '03, pages 669–677, New York, NY, USA, 2003. ACM.

[WZX+16] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

[XYS+16] Ke Xie, Feilong Yan, Andrei Sharf, Oliver Deussen, Hui Huang, and Baoquan Chen. Tree modeling with real tree-parts examples. *IEEE transactions on visualization and computer graphics*, 22(12):2608–2618, 2016.

[YARK15] M. E. Yumer, P. Asente, Mech R., and L. B. Kara. Procedural modeling using autoencoder networks. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pages –. ACM, 2015.

[YBY+13] Yi-Ting Yeh, Katherine Breeden, Lingfeng Yang, Matthew Fisher, and Pat Hanrahan. Synthesis of tiled patterns using factor graphs. *ACM Trans. Graph.*, 32(1):3:1–3:13, February 2013.

[YS12] Qizhi Yu and Anthony Steed. Example-based road network synthesis. In *Eurographics (Short Papers)*, pages 53–56, 2012.

[YWVW13] Yong-Liang Yang, Jun Wang, Etienne Vouga, and Peter Wonka. Urban pattern: Layout design by hierarchical domain splitting. *ACM Trans. Graph.*, 32(6):181:1–181:12, November 2013.

[YYT+11] Lap-Fai Yu, Sai-Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F. Chan, and Stanley J. Osher. Make it home: automatic optimization of furniture arrangement. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pages 86:1–86:12. ACM, 2011.

[YYW+12] Yi-Ting Yeh, Lingfeng Yang, Matthew Watson, Noah D. Goodman, and Pat Hanrahan. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Trans. Graph.*, 31(4):56:1–56:11, July 2012.

[Zie04] Mark Ziegelmann. Constrained shortest paths and related problems, 2004. Dissertation.

[ZJL14] Shizhe Zhou, Changyun Jiang, and Sylvain Lefebvre. Topology-constrained synthesis of vector patterns. *ACM Trans. Graph.*, 33(6):215–1, 2014.

[ZLL13] Shizhe Zhou, Anass Lasram, and Sylvain Lefebvre. By–example synthesis of curvilinear structured patterns. In *Computer Graphics Forum*, volume 32, pages 355–360. Wiley Online Library, 2013.

[ZS84] T. Y. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27(3):236–239, March 1984.

[ZW12] Xiaoyan Zhu and Wilbert E. Wilhelm. A three-stage approach for the resource-constrained shortest path as a sub-problem in column generation. *Comput. Oper. Res.*, 39(2):164–178, February 2012.