KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Inverse Procedural Modeling of Buildings

Anđelo Martinović

Supervisor:
Prof. dr. ir. Luc Van Gool

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering Science

September 2015

Inverse Procedural Modeling of Buildings

Anđelo MARTINOVIĆ

Examination committee:
Prof. em. dr. ir. Yves Willems, chair
Prof. dr. ir. Luc Van Gool, supervisor
Prof. dr. ir. Luc Van Eycken
Prof. dr. ir. Philip Dutré
Prof. dr. ir. Luc De Raedt

Prof. dr. Nikos Paragios
(École Centrale de Paris, France)

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Engineering Science

September 2015

Uitgegeven in eigen beheer, Anđelo Martinović, Kasteelpark Arenberg 10, box 2441, B-3001 Heverlee, BELGIUM (Belgium)

ISBN XXX-XX-XXXX-XXX-X

D/XXXX/XXXX/XX

Acknowledgments

First of all, I want to thank my supervisor, prof. Luc Van Gool, for giving me the opportunity to pursue my PhD, and the freedom to find my own research path. I still remember the 'Welcome aboard' email that felt like being accepted to serve on the USS Enterprise. I would also like to thank my examination committee for their insights and helpful comments: prof. Luc Van Eycken, prof. Luc De Raedt, prof. Philip Dutré, and prof. Nikos Paragios.

A big thank you goes to all my co-authors. To the ETH crew: thank you Julien and Hayko, for the work you invested in our papers together, and for introducing me to the nightlife in Zurich following our long project meetings! To my VISICS crew: thank you, Markus, for your friendship, and for working with me practically since day 1 of my PhD. Your guidance allowed me to find the right direction in research, and your positive attitude helped greatly when we encountered late-night deadlines, bugs in code, weird filenames (fold_ttt) and endless waiting for reviews. Thank you Honza, for the awesome paper that concluded my PhD journey. It would've been much harder without you (that's what she said).

I would also like to thank all of my former and current colleagues in Leuven. Many a time I was saved by their indispensable assistance: be it a theoretical discussion with Marco, an insightful paper recommendation from Rodrigo, or an implementation trick and a stern gaze from Markus Moll upon seeing my code, with each bit I got a step closer to my goal. Marco, it was a pleasure sharing an office with you, and thanks for being the best teammate for Bike&Run! To my DEP teaching assistant comrades: Vincebro, your zebra drawing skills shall be missed; Wim, thanks for allowing me to oversleep some of those 8 AM sessions and for being a reliable BSG player! There are many others I should thank (vaguely in order of appearance): Chris, Hakan, David, Sebastian, Jose, Reyes, Kristof, Kostas, Amir, Rosalia, Stam, pokerface Xu, Jay, Stratis, Bert the Young, Gina, Xuanli... the list goes on! Thank you everyone, for our lunches in Alma (and making me forget about the food quality), for the

deadline nights (and making me forget that I should be tired), for the birthday cakes (and making me remember to bring some myself), coke breaks, laser tag, board games, bike trips, barbecues, beach volleyball, squash, and beers on Oude Markt!

An overwhelmingly big thank you goes to all of my friends: the ones back home, the ones in Leuven, and the ones that are now all over the world pursuing their dreams. Thank you Ivan, for being a great friend and an ideal flatmate for almost four years. Your guitar playing was quite tolerable. And thanks for recommending Leuven to a confused college graduate who didn't know where to do his PhD 5 years ago. Thank you Ivana, for the awesome trips across the entire continent, and for being there during the toughest parts of my PhD where you probably learned more about computer science than you ever expected to. A big thanks goes to my friend Boran, who very much enjoys cherries, for all the pancakes over the years. But I still don't admit your victory in trešeta. Viruj mi. A whole gallon of thank you goes to Dušan and Ivana, for all the coffee breaks that kept me sane during long working days, and dinners at your place that created a feeling (and taste) of home. Thank you Andrej, for all the Mišo Kovač parties. Gorana and Ivan, for being great Cylons. Antoine, for surviving a year with me in the apartment. Tomislav and Ivan, niste vi meni svatko! The list of people to thank goes on: Irma, Bogi, Mire, Ivan, Tićma, Vukov, Pajić, Bane and his family...

To my friends from back home (but not necessarily still there), Danko, Jelena, Katja, Boris, Srda, Anđelka, Pećo, Maja, Šokre, Lika, Zaher (cimiiii), Mećko, Joško, Gogo sreća, Čikito, Davor, Irena and Viki: thanks for being there when I needed somebody to talk to, or for letting me crash at your place when I wanted to visit the homeland, or letting me show you around Belgium; for salsa parties, for board games that would last until the next morning (and the evening after), for picigin and trešeta, and for all the summers that recharged my batteries!

And finally, but most importantly, I want to thank my family, from the bottom of my heart. Even though I did not become a waiter in a restaurant as my grandma wished ('Vrag pone i skule'), I'm still well fed. Don't you worry, baba! To my mom and dad, thank you for guiding me all these years, for the sacrifices you made to provide me with opportunities you yourself didn't have, for supporting me when I felt blue, and encouraging me to always strive for more. Mama i tata, hvala vam od srca! This thesis is dedicated to you.

Anđelo Martinović
Leuven, September 2015

Abstract

3D city modeling is a thriving area of research, as high quality models of real-world cities are in ever-rising demand. These models are used not only among architects and urban planners, but also find their application in navigation, virtual tourism, and entertainment industry. Manual modeling of individual buildings usually provides good results, but the process is very time consuming and expensive. Current automatically-built models using Structure from Motion, followed by simple plane fitting and texturing are a good starting point, but provide inadequate 3D visual perception. No matter the capturing technology, the resulting models are deficient, particularly when dealing with thin objects, fragmented volumes and reflective surfaces. Furthermore, conventional bottom-up models lack any semantic knowledge about the scene. Yet, adding a good understanding of what it is that needs to be modeled is a strong cue, not only to improve the visual and 3D quality of the model, but also to substantially widen its usage.

Conversely, procedural modeling provides an effective way to create detailed and realistic 3D building models that do come with all the semantic labels required. This elegant yet powerful framework represents models such as buildings through instantiations of a series of parameterized rules, forming a grammar. The resulting procedural models are compact, rich in terms of semantics, and allow for more aesthetic rendering than would be possible from pure 3D capturing.

Thus far, procedural modeling has largely been used for synthesizing virtual buildings. In this thesis, we investigate how procedural models can be used in the context of urban reconstruction. Our ultimate goal is to automatically create procedural models of structures as-built, a process referred to as inverse procedural modeling. The main challenge in this process is to determine the appropriate rules of the procedural grammar and their parameters, which typically results in a large search space.

In the first part of the thesis we assume the grammar rules are already known, while parameters are allowed to vary. We develop a system for 3D building reconstruction where the grammar leads the modeling process and receives structural information from object detectors. A drawback of this approach is that the grammar needs to be selected beforehand. Therefore, we develop an algorithm for automatic selection of the appropriate style grammar based on the visual recognition of the architectural style of the observed building.

Presently, the main drawback of procedural grammars is that expert architectural knowledge is needed for their creation, which is a non-trivial manual process. Moreover, the abundance of different architectural styles in the world would require many such grammars to be designed. To tackle this problem, in the second part of the thesis we simplify the prior knowledge necessary for building reconstruction to a set of general and style-independent architectural principles. We use these weaker priors in a bottom-up approach, by producing high-quality semantic labeling of perspective images and Structure-from-Motion point clouds. This labeling is afterwards transformed into building-specific procedural models, allowing realistic rendering.

In the third part of the thesis, we address the problem of procedural grammar scarcity by proposing to learn the grammars from data. First, we show that probabilistic grammars can be learned from annotated facade imagery. The inferred grammars are shown to be comparable to expert-written grammars on the task of facade reconstruction. Second, we eliminate the need for manual image annotation by replacing it with the previously proposed automatic facade labeling approach. Finally, the learned representations are shown to be useful for virtual facade synthesis, facade comparison and retrieval.

The proposed models have been evaluated on several datasets of urban scenes, advancing the state of the art in terms of accuracy and speed. More importantly, it is the conclusion of this thesis that the problem of inverse procedural modeling of buildings could be solved with grammar learning from labeled and noisy data, obviating the need for a human in the loop, and opening up novel directions for future research.

Beknopte samenvatting

3D stadsmodellering is een bloeiend gebied van onderzoek, omdat er een steeds groeiende vraag is naar stadsmodellen van hoge kwaliteit. Deze modellen worden niet alleen gebruikt bij architecten en stedenbouwkundigen, maar vinden ook hun toepassing in navigatie, virtueel toerisme en de entertainment industrie. Handmatige modellering van individuele gebouwen levert meestal goede resultaten op, maar het proces is zeer tijdrovend en duur. Huidige modellen, automatisch gebouwd met behulp van structuur-uit-beweging, gevolgd door eenvoudige plane fitting en textuurbekleding zijn een goed uitgangspunt, maar bieden onvoldoende visuele kwaliteit. Ongeacht de manier van acquisitie hebben de resulterende modellen een tekort, vooral wanneer het gaat om dunne voorwerpen, gefragmenteerde volumes en reflecterende oppervlakken. Bovendien missen conventionele bottom-up modellen semantische kennis over de scene. Maar een goed begrip van wat moet gemodelleerd worden is een sterke hulp, niet alleen om de visuele en 3D kwaliteit van het model te verbeteren, maar ook om het toepassingsgebied aanzienlijk te vergroten.

Procedurele modellering biedt een effectieve manier om gedetailleerde en realistische 3D-modellen van gebouwen te creëren die wel alle vereiste semantische labels meekrijgen. Dit elegante maar krachtige framework representeert modellen zoals gebouwen door de instantiatie van een reeks geparametriseerde regels, die samen een grammatica vormen. De resulterende procedurele modellen zijn compact, rijk in termen van semantiek, en zorgen voor meer esthetische rendering dan mogelijk zou zijn door pure 3D vastlegging.

Tot dusver werd procedurele modellering grotendeels gebruikt voor het synthetiseren van virtuele gebouwen. In dit proefschrift onderzoeken we hoe procedurele modellen kunnen worden gebruikt in het kader van de stadsreconstructie. Ons uiteindelijke doel is om procedurele modellen van bestaande structuren automatisch te creëren, een proces dat bekend staat als inverse procedurele modellering. De belangrijkste uitdaging in dit proces is het bepalen van passende regels voor de procedurele grammatica en hun parameters,

wat typisch resulteert in een grote zoekruimte.

In het eerste deel van het proefschrift nemen we aan dat de grammaticale regels reeds bekend zijn, terwijl de parameters mogen variëren. We ontwikkelen een systeem voor 3D reconstructie van gebouwen waar de grammatica het modelleringsproces leidt en structurele informatie ontvangt van object detectoren. Een nadeel van deze benadering is dat de grammatica vooraf moet worden geselecteerd. Daarom ontwikkelen we een algoritme voor automatische selectie van de juiste stijlgrammatica op basis van visuele herkenning van de bouwstijl van het waargenomen gebouw.

Momenteel is het belangrijkste nadeel van procedurele grammatica dat deskundige architecturale kennis vereist is om er een te creëren, wat een niet-triviaal handmatig proces is. Bovendien zou de overvloed aan verschillende architecturale stijlen in de wereld ervoor zorgen dat veel van dergelijke grammatica's moeten worden ontworpen. Om dit probleem aan te pakken vereenvoudigen we in het tweede deel van het proefschrift de voorkennis die nodig is voor gebouwsreconstructie tot een reeks algemene en stijl-onafhankelijke architecturale principes. We gebruiken deze zwakkere voorkennis in een bottom-up benadering, door het produceren van kwalitatief hoogwaardige semantische labels van perspectiefbeelden en structuur-uit-beweging puntenwolken. Deze labels worden daarna omgezet in gebouw-specifieke procedurele modellen, waardoor realistische weergave mogelijk is.

In het derde deel van het proefschrift pakken we het probleem van de schaarste van procedurele grammatica aan door de grammatica uit data te leren. Eerst tonen we aan dat probabilistische grammatica's kunnen worden geleerd uit geannoteerde beelden van facades. We tonen dat de afgeleide grammatica's vergelijkbaar zijn met grammatica's geschreven door experts voor de taak van facade reconstructie. Ten tweede elimineren we de noodzaak voor handmatige beeldannotatie door deze te vervangen met de eerder voorgestelde automatische facade labeling aanpak. Tenslotte wordt aangetoond dat de geleerde representaties nuttig zijn voor de synthese van virtuele facades, de vergelijking van verschillende facades en het vinden van facades.

De voorgestelde modellen zijn geëvalueerd op verschillende datasets van stedelijke scènes en bevorderen de state of the art op het gebied van nauwkeurigheid en snelheid. Wat nog belangrijker is, het is de conclusie van dit proefschrift dat het probleem van de inverse procedurele modellering van gebouwen kan worden opgelost door het leren van grammatica uit gelabelde en ruizige gegevens, wat de noodzaak van een menselijke ingreep wegneemt en nieuwe richtingen voor toekomstig onderzoek openstelt.

Kratak sažetak

3D modeliranje gradova je vrlo aktivno polje istraživanja, budući da postoji sve veća potražnja za visokokvalitetnim modelima postojećih gradova. Osim kod arhitekata i dizajnera urbanih prostora, dotični modeli pronalaze svoju primjenu i u navigaciji, virtualnom turizmu i industriji zabave. Ručno modeliranje pojedinačnih građevina obično rezultira kvalitetnim modelima, ali zahtijeva puno vremena i novca. Trenutni modeli generirani automatskim postupcima poput "Strukture iz pokreta" (engl. Structure from Motion, SfM), nakon kojeg slijedi određivanje ravninskih segmenata i teksturiranje su dobra početna točka, no 3D percepcija takvih modela je nezadovoljavajuća. Koju god tehnologiju za prikupljanje podataka koristili, stvoreni modeli su manjkavi, pogotovo kad sadržavaju tanke predmete, rascjepkane volumene ili reflektirajuće površine. Povrh toga, konvencionalni modeli oskudijevaju semantičkim znanjem. Međutim, ukoliko je struktura objekta koji se modelira dobro poznata, moguće je ne samo poboljšati vizualnu i 3D kvalitetu modela, već i znatno proširiti njegovu uporabu.

S druge strane, proceduralno modeliranje pruža učinkovit način za stvaranje detaljnih i realističnih 3D modela građevina, koje dolaze sa svim potrebnim semantičkim oznakama. Ovaj elegantni no snažni programski okvir predstavlja modele poput građevina u algoritamskom obliku. Svaki model je opisan gramatikom, koja sadrži skup parametriziranih produkcija. Počevši od jednostavnog modela, sekvencijalnom primjenom produkcija model se razrađuje i obogaćuje dodatnim detaljima. Rezultirajući proceduralni modeli su kompaktni, semantički bogati, i omogućuju realističnije renderiranje modela.

Do sada, proceduralno modeliranje se uglavnom koristilo za sintezu virtualnih građevina. Ova disertacija istražuje načine na koje se proceduralni modeli mogu koristiti u kontekstu rekonstrukcije urbanih prostora. Krajnji cilj disertacije je automatsko stvaranje proceduralnih modela iz postojećih struktura, proces poznat kao inverzno proceduralno modeliranje. Glavni izazov ovog postupka je određivanje odgovarajućih produkcija i njihovih parametera, što tipično rezultira velikim prostorom pretraživanja.

U prvom dijelu disertacije pretpostavlja se da je struktura gramatike poznata, dok parametri produkcija mogu varirati. Predstavljen je sustav za 3D rekonstrukciju građevina gdje gramatika vodi proces modeliranja, primajući informacije o strukturi od detektora objekata. Mana ovakvog pristupa je potreba za odabirom ispravne gramatike prije samog postupka rekonstrukcije. Stoga, razvijen je dodatni algoritam za automatski odabir odgovarajuće stilske gramatike, na temelju vizualnog prepoznavanja arhitektonskog stila građevine.

Trenutno, glavni nedostatak proceduralnih gramatika je činjenica da njihovo stvaranje nije trivijalan proces, i zahtijeva ekspertno poznavanje domene. Osim toga, obilje različitih arhitektonskih stilova u svijetu bi zahtijevalo ručno kreiranje mnogo takvih gramatika. Kao jedan način rješavanja ovog problema, drugi dio disertacije koristi pojednostavljeni model a priori znanja potrebnog za rekonstrukciju građevina. Umjesto cjelovitih gramatika, koristi se skup općenitih pravila neovisnih o stilu građevine. U pristupu odozdo-prema-gore, provedena je visokokvalitetna semantička segmentacija fotografija ili oblaka 3D točaka produciranih SfM pristupom. Ova segmentacija se kasnije može pretvoriti u proceduralni model specifičan za određenu građevinu, omogućavajući realistično renderiranje.

U trećem dijelu disertacije predstavljeno je rješenje za problem oskudice proceduralnih gramatika kroz njihovo učenje iz podataka. Pokazano je da stohastičke gramatike mogu biti automatski naučene iz skupa anotiranih fotografija fasada građevina. Naučene gramatike se pokazuju jednako kvalitetne za rekonstrukciju građevina kao i ručno napisane gramatike. Nakon toga, eliminirana je potreba za ručnom anotacijom fotografija kroz korištenje ranije navedene metode za automatsku semantičku segmentaciju. Konačno, naučene reprezentacije fasada koriste se za sintezu novih, virtualnih zgrada, usporedbu fasada i dohvat sličnih fotografija iz baze podataka.

Predstavljeni modeli su evaluirani na nekoliko skupova podataka, i pokazuju rezultate iznad performansi trenutno najboljih modela u svojim domenama, u pogledu preciznosti i brzine. Zaključak ove disertacije jest da je problem inverznog proceduralnog modeliranja rješiv kroz primjenu učenja gramatika iz podataka, uklanjajući potrebu za ljudskom intervencijom, te ujedno otvara nove pravce budućeg istraživanja.

List of Abbreviations

2D-ASCFG	Two-Dimensional Attributed Stochastic Context-Free Grammar
AdaBoost	Adaptive Boosting
ARC3D	Automatic Reconstruction Cloud
BMM	Bayesian Model Merging
CAD	Computer-Aided Design
CGA	Computer Generated Architecture shape grammar
CKY	Cocke–Younger–Kasami parsing algorithm
CMC	Cumulative Match Characteristic
CNF	Chomsky Normal Form
CRF	Conditional Random Field
CVX	Matlab Software for Disciplined Convex Programming
DP	Dynamic Programming
DPM	Deformable Part-based Model
ECP	École Centrale Paris
EM	Expectation-Maximization algorithm
FLANN	Fast Library for Approximate Nearest Neighbors
FPPI	False Positive Per Image
GA	Genetic Algorithm
IOU	Intersection over Union
IPM	Inverse Procedural Modeling
IQP	Integer Quadratic Program
ISM	Implicit Shape Models
LHS	Left-hand side of a grammar production
LIDAR	Light Radar
LOG	Multiclass Logistic Regression Classifier
MAA	Maximum Achievable Accuracy
MAP	Maximum a posteriori
MCMC	Markov Chain Monte Carlo
MDL	Minimum Description Length
MILP	Mixed-Integer Linear Program
MLP	Multi-Layer Perceptron
MOSEK	Large scale optimization software
MRF	Markov Random Field

MVS	Multi-View Stereo
NBNN	Naive Bayes Nearest Neighbor
NN	Nearest Neighbor
PASCAL	Pattern Analysis, Statistical modeling, and Computational Learning
PCL	Point cloud
PMVS	Patch-based Multi-View Stereo
RANSAC	RANdom SAmpling Consensus
RF	Random Forest
RHS	Right-hand side of a grammar production
rjMCMC	Reversible Jump Markov Chain Monte Carlo
RL	Reinforcement Learning
RNN	Recursive Neural Network
SCFG	Stochastic Context-Free Grammar
SEEDS	Superpixels Extracted via Energy-Driven Sampling
SfM	Structure from Motion
SI	Spin Image
SIFT	Scale-Invariant Feature Transform
SNF	Simple Normal Form
SSIM	Self-Similarity Descriptor
SVM	Support Vector Machine
VOC	Visual Object Classes
WP	Weak architectural Principles

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A Brief History of City Modeling

Attempts at urban space modeling can be traced back as early as 1400 BC. In the Sumerian city of Nippur, a clay tablet was discovered containing the oldest known city map, depicting the labeled points of interest in the ancient city, such as the city park, walls, and the river Euphrates.

Yet, there are very few written sources from the classical ages about the theory of architecture or city planning that have survived to this day. The only surviving written source on classical architecture is the famous work of Vitruvius, *De architectura*. In his treatise, Vitruvius compiled the sum of knowledge on Roman building methods, including notes on town planning, construction of temples, civil and domestic buildings. Although the original ancient illustrations were lost, the detailed instructions in the original text allowed historians and architects to recreate the models according to their interpretations (see Fig. 1.2).

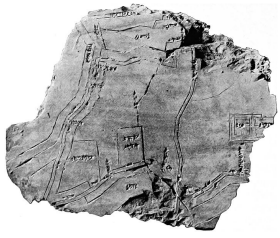


Figure 1.1: Ancient map of the Mesopotamian holy city, Nippur, circa 1400 BC.

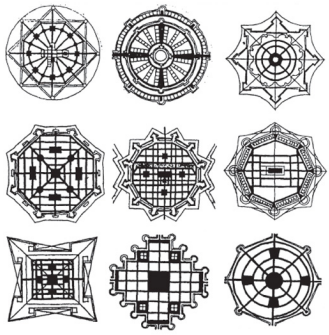


Figure 1.2: Ideal city plans from various Renaissance authors, as inspired by Vitruvius.

Figure 1.3: Aerial view of the Italian city of Palmanova, built in 1593. according to Vitruvius' plans.

Figure 1.4: A depiction of Vienna, from Nuremberg Chronicle, 1493.

In the Late Middle Ages, city maps and illustrations had both a topographical and artistic purpose. Since the theory of perspective projection was not developed until Renaissance, cities were mostly depicted as orthographic plans, or as seen from profile (See Fig. 1.4). Many such examples are compiled in one of the most comprehensive collections of city illustrations of that age, the Nuremberg Chronicle.

In Renaissance, artists and architects became interested in how to create realistic depictions of landscapes, urban spaces and buildings so that they look the same as in real world. The solution in the form of linear perspective was brought forth by Filippo Brunelleschi, the famous architect and creator of the Florence Cathedral dome. Consequently, cartographers and artists started using the new technique to create geometrically exact 3D perspective views of cities,

Figure 1.5: A perspective view of Antwerp, from the *Civitates orbis terrarum* atlas, Braun and Hogenberg, 1572.

Figure 1.6: St. Blaise holding the model of the city of Dubrovnik, Nikola Božidarević, beginning of the 16th century.

see Fig. 1.5.

An interesting example of a 'virtual' city model appearing in a Renaissance painting is that of the city of Dubrovnik, Croatia. In a tryptych on a side altar in the Dominican monastery in Dubrovnik, the saint protector of the city, St. Blaise, is depicted holding a 3D model of the city in his hands, see Fig. 1.6.

1.2 City Modeling Today

The advent of technology has allowed us to create maps and models of cities in ways never before imagined. These models find their application in several fields, such as digital mapping, urban planning, archaeology, and entertainment industry.

1.2.1 Digital Mapping

In recent years, tech giants like Google, Microsoft and Apple have engaged in a race of providing high-quality city models to the broad public. Initial models produced by their methods were quite rudimentary. For example, in the early versions of Google Earth software, cities were modeled as satellite images pasted on the 3D model of the globe, followed by simple extrusions of building outlines. Afterwards, users were encouraged to manually create their own building models in a sketching software, such as Google Sketchup. The

Figure 1.7: A Google Earth model of Dubrovnik with user-created models in Sketchup, 2011.

sketching approach opened up the possibility of crowdsourcing the city modeling task to millions of people, even non-experts, but it also had several drawbacks. For example, only some buildings were modeled (usually the attractive or popular landmarks), and the quality of each model differed depending on the invested modeling time or the skill of the modeler. As an illustration, Fig. 1.7 shows the incomplete model of Dubrovnik as it appeared in Google Earth in 2011.

1.2.2 Urban Planning

Accurate 3D city models on varying levels of abstraction can be utilized to represent the current state of an existing urban environment. These models can then be used by the city administration to e.g. manage spatial regulatory rules, analyze the urban energy efficiency, determine the solar power potential of roofs, plan the placement of wireless access points, etc. The creation of these models is often assigned to specialized companies which provide services of airborne or ground-based surveys combined with manual GIS modeling, as exemplified in Fig. 1.8.

1.2.3 Archaeology

Thanks to the rapid development of computer graphics and imaging, archaeologists can now use computer software to model and visualize

Figure 1.8: A 3D building model of Dubrovnik based on LiDAR laser scans. The model is in accordance with the Level-of-Detail 2 in the CityGML standard. Image courtesy GDi GISDATA, 2012.

Figure 1.9: A blended image of the reconstruction and a photograph of the second Temple of Hera, c. 460-450 BC, Paestum, Italy. See Chapter 4 for more details on the automatic reconstruction process.

archaelogical sites in 3D. The 3D models of archaelogical sites created by experts in the field facilitate scientific discussion about reconstruction hypotheses, and can also be used in education, entertainment and site marketing. Some examples of these methods include Rome Reborn [89], an initiative to create a digital model of ancient Rome; 3D reconstruction of Mayan buildings in Xkipche [124], and modeling of Greek Doric temples [116], see Fig. 1.9.

Figure 1.10: City modeling in the movie industry: a CGI-enhanced image of Dubrovnik represents the fictional city of King's Landing in Westeros. Image courtesy: Mackevision Medien Design GmbH, 2014.

1.2.4 Entertainment Industry

Contemporary computer games based on the open-world paradigm sport complete virtual cityscapes, often inspired by real cities. In movie industry, a common practice is to augment the images or videos of existing cities with computer-generated imagery (CGI). As an example, a city in a popular TV show 'Game of Thrones' was based on Dubrovnik, see Fig. 1.10. The created models are however based on manual work of a whole team of visual effect designers, utilizing expert tools such as Maya or 3D Studio Max.

1.3 Data Acquisition

Two most popular methods for obtaining the data from urban scenes are imagery and LiDAR.

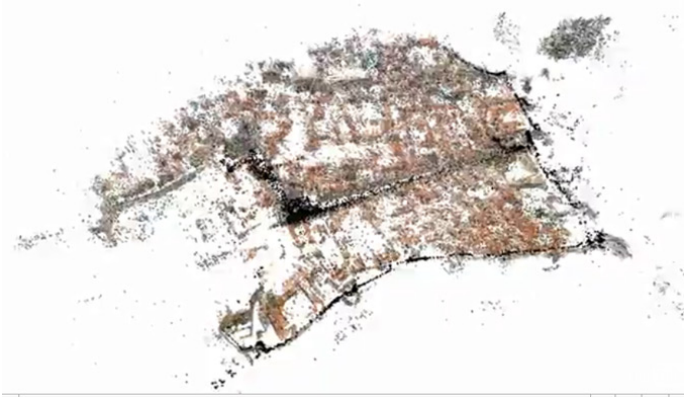


Figure 1.11: A point cloud model of the city of Dubrovnik automatically created from 58k tourist photographs [2], 2009.

1.3.1 Imagery

Due to the widespread use of digital cameras nowadays, vast amounts of photographs are being created each year[1]. Many of these photographs are available on the Internet, a large portion of them depicting urban sites. Various tools have been developed that harvest this source of information. For example, the project 'Building Rome in a Day' [2] mines images from popular touristic destinations, such as Rome, Venice or Dubrovnik, and creates 3D models from the automatically downloaded data, see Fig. 1.11.

On the other hand, aerial and satellite imagery are becoming more and more popular thanks to web-mapping projects, such as Apple Maps, Bing Maps, and Google Earth. The techniques of aerial photogrammetry have advanced to the point where they can be used to automatically produce 3D city models, complete with trees, landmarks and realistic textures.

1.3.2 LiDAR

As an alternative to image-based methods, it is possible to use LiDAR[2], i.e. 3D laser scanning. Although LiDAR provides relatively precise data in form of semi-dense 3D point clouds, it has its drawbacks, such as the inability to

[1] According to the survey of [129], an estimate of tens of billions of photos are taken worldwide each year.

[2] Abbreviation of Light Radar, although sometimes interpreted as Light Detection and Ranging.

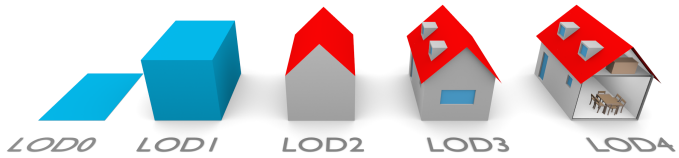


Figure 1.12: The concept of 5 Levels of Detail (LOD) as defined in CityGML 2.0. Image from [13].

capture colour information and bad performance on transparent surfaces. More importantly, the relatively high cost of LiDAR devices limits its application to land surveying offices or civil engineering bureaus.

Some approaches use both sources of data (images and LiDAR) to combine the high resolution of image data with the more precise 3D structure obtained from LiDAR data. For example, at the time of writing of this text, Nokia is using a fleet of 300 cars equipped with LiDAR and high-resolution cameras to map major cities around the world. In the future, the produced models could be used by driverless cars to navigate roadways.

1.3.3 Terrestrial vs. Aerial

Both image-based and LiDAR surveying systems can be mounted on terrestrial or aerial platforms. These platforms consist of a set of cameras (or a LiDAR sensor, or a combination of the two), a GPS receiver, an inertial measurement unit (IMU), an onboard computer and data storage devices.

Different platforms are suitable for urban models with varying level of detail (LOD). According to the OpenGIS CityGML standard [71], airborne data is more suitable for coarse building modeling (LOD1 and LOD2), as it captures the building in its entirety, including roofs. However, its limited resolution on the facade and street level calls for the use of terrestrial data when it comes to individual building and facade modeling (LOD2 and LOD3). See Fig. 1.12 for an illustration of different LOD models in CityGML.

1.4 Limitations of Existing Methods

Current automatic urban reconstruction methods are far from being perfect. Human intervention is typically needed to remove visible artifacts, correct textures or fill in incomplete data. Errors in the reconstruction appear regardless

of the capturing technology. For example, image-based methods perform poorly on non-textured surfaces or in presence of significant variation in illumination, while LiDAR encounters problems when dealing with transparent surfaces.

Yet, strong priors can be used as soon as one knows what it is that one is trying to model. Knowledge about the semantics of the scene can not only aid the reconstruction process, but also allow for new applications of generated models. For example, one could simulate living cityscapes with crowds of people entering buildings through doors, looking through windows, using public transport, etc. Semantic models also enable data mining queries, for example, calculating the combined surface area of all windows in a city.[3]

1.5 Requirements

We identify the following 5 requirements needed for a comprehensive system for semantically-driven urban reconstruction:

- R1. The system must be able to model different types of buildings, architectural styles, landmarks etc.
- R2. The system should use a consistent representation (model) of buildings that is rich in semantics, allows easy rendering, model retrieval, and novel model design.
- R3. If expert-designed building models are available, the system should be able to use these models to reconstruct existing buildings.
- R4. In the cases when no models are available, or we are dealing with an atypical building, the system should perform reconstruction with a bottom-up approach.
- R5. The system should be able to learn from experience, i.e. to use previously reconstructed buildings to help guide the process in the future.

In recent years, *procedural modeling*, a technique from the field of computer graphics, has emerged as a powerful method to model urban spaces. Procedural modeling describes 3D models of objects through instantiations of a series of rules. It allows the modeler to create simple textual representations, rule-based descriptions of complex models like plants, facades, buildings, or street networks.

[3] Interestingly, a similar question, 'How much would you charge to wash all the windows in Seattle?' is allegedly posed to candidates applying for a job at companies such as Amazon and Google, aimed at evaluating their estimation skills.

In addition, procedural models are rich in terms of semantics. Semantic concepts like windows, floors, doors, balconies, etc. are made explicit in the ruleset. Procedural city models can therefore be explored at a high semantic level. Moreover, these models are typically parametric, so by changing the parameters of a single model, a large number of variations can be created. This makes procedural modeling an ideal candidate to satisfy requirements R1 and R2.

The next question is if procedural modeling can be used to reconstruct existing buildings. This is the topic of *inverse procedural modeling*, i.e. discovering the procedural model that fits the real-world observations. One method of doing this would be to start from a parametric procedural model, and then attempt to find the optimal set of parameters such that the resulting model resembles the observations as much as possible. This satisfies requirement R3, as a reconstruction of the existing building can be posed as a parameter estimation problem.

Ideally, not only the parameters, but the procedural rules themselves would be inferred from real-world observations (R5). However, there is a large semantic gap between images and procedural grammars, which makes grammar learning directly from images a very challenging problem. In order to close this gap, a bottom-up approach for semantic analysis of images would prove useful, not only for reconstruction of atypical buildings (R4) but also to bootstrap the model learning procedure (R5).

Having identified the requirements and the means of satisfying them, it becomes clear that this thesis is going to be positioned on the intersection of computer vision and computer graphics. Since procedural models are typically represented with formal grammars, we will also venture into the domain of formal language theory, and come into contact with the field of natural language processing, especially when encountering problems such as grammar parsing and inference.

1.6 Motivation

Motivated by the requirements from the previous section, we formulate five main research questions which are the topic of this thesis:

- RQ1. What kind of procedural models are useful for the task of city-scale reconstruction?
- RQ2. How can we fit an appropriate procedural model to observed data in an efficient way?

- RQ3. How can we reconstruct buildings when no procedural model is available?
- RQ4. Is it possible to learn the procedural rules from data in a supervised way, assuming there is no noise in the training data?
- RQ5. Can we learn procedural models even from noisy data?

To address the question RQ1, we investigate the procedural grammars commonly used in architectural modeling, paying attention to their usability for forward modeling, simplicity, and the possibility of inference. The generalization capabilities of different grammars have a high impact on the obtained results. A grammar that is too general does not introduce enough constraints, which keeps the search space too large. On the other hand, a grammar that is too specific might be applicable to a very small subset of buildings, thus making it less useful for large scale reconstruction.

As will be shown later in the text, our intuition is that one particular procedural grammar roughly corresponds to an architectural style. Thus, to select the right procedural model for each building, we investigate the problem of architectural style classification (R1 and RQ2). We investigate different ways of fitting the selected grammars to a certain building, either by direct estimation of grammar parameters, or exploration of the parameter space (R3 and RQ2).

To address R4 and RQ3, we investigate bottom-up approaches for semantic segmentation of buildings and facades. In this case, when procedural models are not available from the outset, we introduce general architectural principles that allow us to express preferences such as symmetry or alignment, which help us in closing the semantic gap between image pixels and semantically meaningful regions of a building, such as windows or balconies. The proposed approach analyzes one building at a time, producing independent building models.

Learning grammar rules from annotated data (R5 and RQ4) is investigated from a Bayesian perspective, following previous success stories in grammar learning in natural language. We pose the grammar inference problem as a search problem driven by a Minimum Description Length principle. This allows us to automatically learn a grammar for a certain building style, given a sufficient amount of clean training data in the form of images annotated with semantic classes.

Finally, we address the problem of structure learning from noisy data (RQ5) as a semi-supervised problem. Instead of relying on supervision in form of manually annotated images, we propose a co-segmentation approach which discovers inherent similarities between facades in the database, while keeping

the structure compatible with existing models of procedural grammars. Some supervision is added through the use of a generic bottom-up classifier (RQ4).

1.7 Contributions

In the following we present the major contributions of this thesis, ordered by their apperance in the text.

1. We present a system to recognize the architectural style of a building, based on low-level features. To make this problem applicable to street-side imagery, we introduce pre-processing steps which rectify the perspective image and separate individual facades.

2. We propose an approach to perform 3D building reconstruction based on an existing procedural grammar. We use Structure from Motion (SfM) and object detectors to find the optimal parameters of the grammar. The approach is demonstrated in the case study of Greek Doric temples.

3. In the case when a procedural grammar is not available, we devise a three-layered approach for semantic segmentation of building facades. We achieve good results by using a combination of strong image region features, object detectors and architectural knowledge. The latter is introduced in the form of novel *weak architectural principles*.

4. We scale the facade segmentation approach to street-side scenes with a large number of images. Starting from an SfM reconstruction, the system separates facades and performs classification purely in 3D, achieving significant speed benefits which allow us to analyze entire streets in a matter of minutes.

5. We propose a Bayesian approach to learn a procedural grammar from a set of annotated facade images. The induced grammar can be used for creating novel instances of the same building style. When applied to semantic segmentation of existing facade imagery, obtained results are comparable to approaches that use a manually designed grammar.

6. Finally, we present an approach that learns structural facade representations even from noisy data. Instead of relying on image annotation, we use a co-segmentation approach that utilizes low-level image features and labeled images from a semantic classifier. We show that the approach produces consistent hierarchical segmentations, usable for facade retrieval and conversion into procedural grammars.

Other, smaller contributions are introduced in their respective chapters, for instance the 2D Earley parser, which is a building block of our Bayesian grammar learning approach.

1.8 Thesis Outline

This thesis is structured as follows. In Chapter 2 we introduce some of the basic concepts used in the thesis, such as formal grammars and procedural modeling.

Chapter 3 deals with prerequisite steps needed for grammar-based urban reconstruction. In large scale applications, such as driving a mobile mapping system through a city, many different architectural styles may be observed. For each style, a different grammar might be available, and the system must decide which grammar to load. Therefore, we propose a system to classify the architectural style based only on image features. Furthermore, the system also filters images containing clutter, rectifies the input perspective images to a fronto-parallel view, and performs separation of individual facades.

In Chapter 4 we perform 3D building reconstruction using an expert-written grammar for Greek Doric temples. In this case, the final reconstruction relies strongly on the meta-knowledge encoded in the rules of the procedural grammar. The rules are enriched with parameters such as the size of the temple, height of its columns etc. We propose to estimate these parameters directly through the combination of structure from motion (SfM) and adaptive object detectors. Finally, building reconstruction is performed by instantiating the procedural model with the learned parameters.

In Chapter 5 we propose a bottom-up approach for semantic segmentation of building facades. The system is organized in three distinct layers, representing increasing levels of abstraction. Region-based semantic segmentation from the first layer is combined with object detectors with a Conditional Random Field formulation in the second layer. The third layer introduces architectural knowledge, but unlike grammar-based approaches, we use much weaker priors, in the form of intuitive architectural principles, such as symmetry, similarity or alignment. Finally, we use the resulting semantic labeling of the facade image to create a realistic instance-specific procedural model, enriched with features such as protruding balconies and reflective windows.

Chapter 6 generalizes the previous approach to three dimensions, starting from a point cloud of a street-side scene obtained by running standard SfM techniques. We then perform classification and facade separation directly in 3D, instead of relying on the original perspective images. This approach results in high-

quality scene labeling with tremendous speed improvements. Further quality improvements may be obtained by complementing the approach with its 2D counterpart. Finally, we adapt the weak architectural principles to 3D, and propose a more efficient optimization scheme based on integer programming.

In Chapter 7 we present an approach for automatic induction of a specific kind of procedural grammars from annotated data. Specifically, we investigate how two-dimensional stochastic context-free grammars can be inferred from a set of ground-truth labelings of building facades. To accomplish this, we adapt a technique from the field of natural language processing, namely Bayesian model merging. We show that the induced stochastic grammar has multiple uses. First, we can sample designs from the grammar to produce novel facades similar to ones in the training set. Second, the grammar can be used to semantically segment existing images of facades outside of the training set. Facade parsing based on the learned grammar is shown to be on par with approaches that use an expert-designed procedural grammar.

Chapter 8 introduces an approach for automatic discovery of high-level structural representations of building facades, with no need for ground-truth annotations. Instead, we propose a joint analysis of a set of facade images and their bottom-up semantic segmentations. We use the idea of co-segmentation to produce consistent segmentations, and propose to use graph clustering to identify semantically similar parts of facades. The process is performed recursively, resulting in rectilinear subdivisions which can be converted to procedural grammars to peform virtual facade synthesis, or used directly for applications such as facade retrieval.

Finally, in Chapter 9 we conclude the thesis with some final remarks, presenting an outlook and possible future work.

Chapter 2

Background

2.1 Procedural Modeling

Procedural modeling is a generic term that applies to all approaches that use rewriting rules to generate geometry. Among them, L-systems, generative modeling and shape grammars are one of the most popular approaches which find their application in city modeling. Both are subtypes of more general models known as formal grammars.

2.1.1 Formal Grammars

In the 1950s, Noam Chomsky attempted to define the syntax of natural languages using precise and simple mathematical rules. In [28], the knowledge of language is modeled using a formal grammar, which contains a set of production rules that rewrite strings, i.e. turn one string into another. A formal grammar is defined as a tuple $G = (V, T, S, P)$, where

- V is a finite, non-empty set of non-terminal symbols.
- T is a finite, non-empty set of terminal symbols, disjoint from V.
- $w \in V$ is the start symbol (sometimes called the axiom, the initial, or the sentence symbol).
- P is a finite set of productions (rules) of the form $\alpha \to \beta$ where $\alpha \in (V \cup T)^* V (V \cup T)^*$ and $\beta \in (V \cup T)^*$. Here, $*$ denotes the Kleene star.

Thus, the left-hand side (LHS) α of a production is a string that contains at least one non-terminal, while the right-hand side (RHS) β is a (possibly empty) string of terminals and non-terminals.

The generation of a string in a language begins from a string containing only the start symbol. The productions are then applied one after another until the resulting string contains only terminal symbols. Each production rewrites a part of the string equal to the LHS of the production with the RHS of the production. The sequence of applied productions is called a **derivation**, while the set of all strings that can be generated by a formal grammar G is called a formal **language**, denoted as $L(G)$. Note that a grammar is a finite object that can describe a potentially infinite language.

Chomsky organized formal grammars into four classes, by gradually increasing the restrictions on the form of productions, a classification today known as the Chomsky hierarchy [27]:

- Type-0 grammars are **unrestricted**, i.e. they may contain rules that transform an arbitrary non-zero number of symbols into an arbitrary (possibly zero) number of symbols.
- Type-1 grammars can be either **monotonic** or **context-sensitive**. A grammar is monotonic if each production satisfies $|\alpha| \leq |\beta|$, while it is context-sensitive if all productions are of the form $\gamma\alpha\delta \to \gamma\beta\delta$. A special rule allows Type-1 grammars to contain rules of the form $S \to \epsilon$, where S may not appear on the RHS of any rule, and ϵ is an empty string. Chomsky proved [29] that monotonic and context-sensitive types of grammars are weakly equivalent, i.e. that they generate the same formal language.
- Type-2 grammars are **context-free**, where each production $\alpha \to \beta$ satisfies $|\alpha| = 1$.
- Type-3 grammars can be either **right-regular** or **left-regular**. A grammar is right regular if each of its productions has one of the following forms:

  $$A \to cB, A \to c, A \to \epsilon,$$

  where A and B are non-terminals (possibly equal), and c is a non-terminal. Conversely, a grammar is left-regular if its productions are of the form

  $$A \to Bc, A \to c, A \to \epsilon.$$

  Right- and left-regular grammars are weakly equivalent.

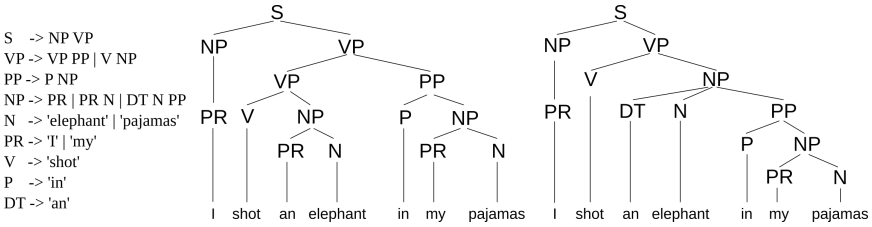


Figure 2.1: "...How he got into my pajamas I'll never know". The famous quote from Groucho Marx is parsed using a simple CFG (left). The symbol '|' denotes a nondeterministic production with multiple choices for the right-hand side. There are two possible interpretations of the sentence as indicated by the two parse trees (who is wearing the pajamas?), illustrating ambiguity in natural language.

Type-(i) grammars include all Type-$(i+1)$ grammars, e.g. all regular grammars are also context-free. The set of Type-0 grammars includes all formal grammars. Finally, since there is a countably infinite number of formal grammars, some languages exist that no formal grammar can generate.

Each of the restrictions on the form of productions reduces the expressive power of a grammar. The resulting grammars are easier to understand but less powerful. But is human readability the only reason to use them? Consider the problem of **parsing**, i.e. recognizing whether a given string can be generated by a certain grammar. Parsing allows us to discover the structure of the input, in certain cases even with presence of incomplete or noisy inputs.

Unfortunately, formal linguistics tells us that the recognition problem cannot be solved for an arbitrary Type-0 grammar [73]. Moreover, even though parsers for general Type-1 grammars can in principle always be constructed, the complexity of these parsers is PSPACE-complete, thus not particularly useful in practice. In contrast, Type-2 and Type-3 grammars are well-researched, and for these grammars many efficient parsing algorithms exist.

Type-3 grammars are typically used in the form of regular expressions, for input matching or to analyze the lexical structure of computer programs. However, they are not expressive enough to capture more complex structures, such as natural language. In fact, Type-2 grammars (Context-free grammars or CFGs) were originally invented by Chomsky to describe the structure of natural language. Yet, their definition turned out to be too restrictive as well, as many natural languages exhibit context-sensitivity [156]. Nevertheless, CFGs are still used in linguistics [139] and in computer science, for example to describe the structure of programming or markup languages, such as XML. The simplicity of productions in a CFG, with only one symbol on the LHS of each production,

Figure 2.2: The Sierpinski triangle constructed with a Sierpinski arrowhead curve L-system.

enables, the parse of a given string to be represented as a tree, called a **parse tree**. See Fig. 2.1 for an example.

2.1.2 L-Systems

Lindenmayer systems (or L-systems for short) are a type of formal grammars, and one of the earliest approaches to procedural modeling. They were named after their inventor, Aristid Lindenmayer, a Hungarian botanist who used these systems to model the growth of plants. Recently, they have found several applications in computer graphics, such as fractal generation and plant modeling.

An L-system is a string rewriting system, defined as a tuple $\mathbf{G} = (V, w, P)$. which is similar to a formal grammar definition, except that terminal and nonterminal symbols are grouped in one The key feature that distinguishes L-systems from other grammars is parallel rewriting, i.e. L-systems apply all productions in parallel. This behaviour corresponds to biological organism growth, where many cell divisions may occur simultaneously.

Although the initial L-system formulation was based on strings, they were soon adapted to allow a graphic interpretation. Prusinkiewicz [137] utilized turtle geometry, where the turtle represents a type of cursor with three attributes: x and y coordinates in the Cartesian plane, and heading (direction in which the turtle is facing). The turtle responds to commands such as moving forward, and turning right or left. This geometric interpretation allows L-systems to model fractals and plants. For example, the Sierpinski triangle as depicted in Fig. 2.2 can be expressed with the following L-system: $(V = \{\text{X,Y}, +, -\}; w = \text{X}; P = \{p_1 : \text{X} \longrightarrow \text{X-Y-X}, p_2 : \text{Y} \longrightarrow \text{Y-X-Y}\})$ where symbols X and Y both represent the command to draw by moving forward one step, while + and - are commands to turn 60 degrees to the left and to the right, respectively.

L-systems were successfully used to model realistic plants and trees [138]. In city modeling, they can also be utilized to generate street maps [135]. Inverse modeling approaches, i.e. discovering the L-system generating a given model, have so far been limited to 2D vector graphics [166].

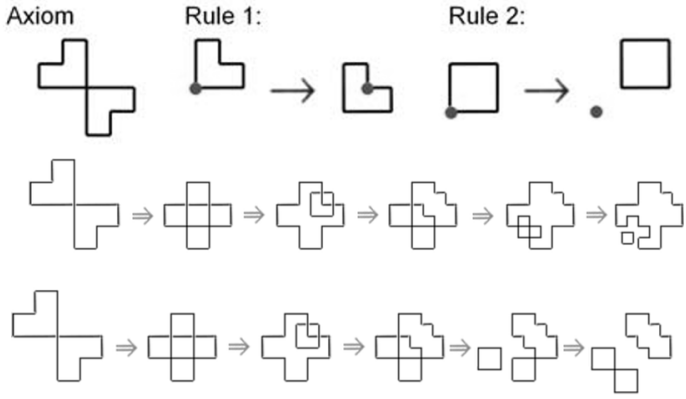


Figure 2.3: An example of a traditional shape grammar. The first row shows the starting shape (axiom) and the two rules the grammar. The second and third row show two possible designs that can be created with this shape grammar.

2.1.3 Shape Grammars

In 1975, Stiny [167] introduced the idea of shape grammars to the field of architecture. The formulation operates on arrangements of lines and points, called shapes. Some points may be labeled by associating a symbol to it. A labeled shape then contains the shape and a set of labeled points. Formally, a shape grammar is defined by four components [168]:

1. S is a finite set of shapes;
2. L is a finite set of symbols;
3. w is the initial labeled shape (axiom) in $(S \cup L)^+$.
4. P is a finite set of shape rules (productions) of the form $\alpha \rightarrow \beta$ where α is a labeled shape in $(S \cup L)^+$ and β is a labeled shape in $(S \cup L)^*$;

The shape rules are applied one at a time to the axiom, or to the shapes previously generated by the rules. A shape rule $\alpha \rightarrow \beta$ can be applied to a labeled shape γ when a transformation τ exists such that $\tau(\alpha)$ is a subshape of γ (subshape problem). The sequence of rule applications is called a derivation. Fig. 2.3 shows an example grammar and its two possible derivations.

The set of all shapes that can be generated by a shape grammar is called a language. Notice that even in this simple example with only two rules, the language can be infinite, and the generated designs quite diverse. Stiny

Figure 2.4: A small facade generated using a split grammar from [202]. The final model is obtained with texturing and inserting pre-modeled elements such as railings.

advocated this liberty of shape grammars, because it supports two important aspects of their expressive power: emergence and ambiguity. On the other hand, the behaviour and design outcomes of the grammar can be hard to control. Thus, the interpretation of shape grammars is usually done manually, or with the assistance of a computer, with a human deciding on which rules to apply.

In order to apply the shape grammar formalism in computer graphics, an automatic shape grammar interpreter is required. This in turn requires solving the subshape problem, i.e. determining all the possible locations in the current shape where the next rule could be applied. This problem was shown to be challenging even for simple grammars, making the problems of shape grammar parsing and inference infeasible [62].

Interest for shape grammars in computer graphics was rekindled when Wonka et al. [202] presented the idea of split grammars, a special type of set grammars [169]. Set grammars circumvent the subshape matching problem by assigning only one symbol to a given shape, thus reducing the problem to simple symbol matching. Elaborating on this idea, split grammars further restrict the set of allowed grammar rules to shape subdivision along a coordinate axis. For example, starting from an initial shape of a building, a split grammar can generate facades of the building, which are then split into their structural elements, repeating the process down to the level of individual design elements like doors, windows, ornaments etc. Wonka also introduced a separate grammar to control the derivation process and to limit the spatial distribution of subshapes in a way that corresponds to architectural principles. Fig. 2.4 shows an example derivation of a small facade using a toy split grammar.

The framework of split grammars was later extended by Müller et al. [125] with the introduction of the CGA shape grammar, which we will detail in the next section.

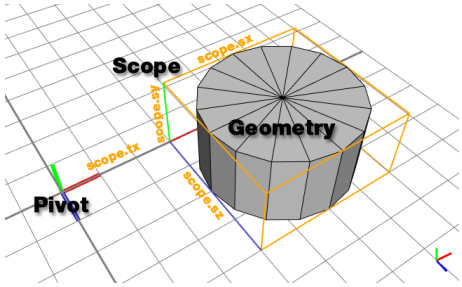


Figure 2.5: The basic attributes of a shape in the CGA shape grammar. Image source: ESRI CityEngine documentation.

2.1.4 CGA Shape Grammar

CGA (Computer Generated Architecture) Shape is a particular type of split grammars created for easy procedural modeling of building mass models [125] and facades [126]. It has since been developed as an integral part of the commercial software CityEngine [42]. In the following, we detail the CGA specification.

Definition

The central element of the CGA grammar is a shape. A shape is identified with a string called the shape symbol, which is used to generate the successors of the shape during the derivation process. Additionally, each shape has associated attributes, which contain the numeric and spatial description of the shape. The most important shape attributes are:

- Geometry, which contains an arbitrary polygonal mesh, color, material and texture information;
- Scope, which represents the oriented bounding box for the shape in space, relative to the pivot;
- Pivot, which describes the coordinate system of the shape, relative to the origin of the initial shape.

Fig. 2.5 illustrates the introduced concepts.

Derivation

Similar to L-systems, the CGA shape grammar operates on a configuration, or a finite set of shapes, and starts the derivation from the axiom. However, unlike L-systems, CGA shape grammar is a sequential system, where only one shape from the configuration is selected in each step of the derivation, and an appropriate rule for replacement is chosen. The derivation stops when the configuration contains no more non-terminal shapes. The derivation tree is explored in a breadth-first manner, first deriving all the shapes in one level before moving to the next level.

Production Rules

The general form of a production rule in CGA is

$$lhs : cond \rightarrow rhs : prob$$

where lhs is a non-terminal shape symbol, which is replaced with the set of symbols rhs, if the condition $cond$ is satisfied. The condition allows context-sensitivity in rule derivation. For example, a window should only be placed if there is enough space on the wall. Stochastic variations of a single model are possible due to the rule selection probability $prob$. The symbols on the right-hand side of the rule rhs can be defined explicitly or through shape operations, defined in the next section.

Operations

Shape operations can create new shapes and alter the current shape. In the following, we summarize some of the most important operations.

- The **insert** operation reads a geometry asset, e.g. a polygon mesh, from a file and inserts it in the scope. It is most commonly used to read atomic elements which should not be generated procedurally, like ornaments.
- The **extrude** operation can create 3D shapes from 2D faces by extruding each face polygon of the current shape in a given direction. For example, it can be used to create a building mass model from a given footprint.
- Conversely, the **component split** operation divides a shape into its geometric components based on a set of semantic selection keywords. For example, it can be used to select faces of the building that are adjacent to a road.

- The **split** operation subdivides the current shape along a specified axis into a set of smaller shapes. It is indispensable in facade modeling, e.g. for splitting the facade into floors, or floors into a set of repeating window tiles. Repeating elements can be modeled by using a * modifier on a sub-shape, see Fig. 2.6 for an example.

Other operations include translation, rotation and scaling of the scope, texturing, specific functions for roof creation etc.

Applications

As mentioned earlier, the CGA shape grammar owes much of its popularity to its implementation in CityEngine [42], a commercial software for procedural modeling. Since their introduction, CGA-based procedural models have been successfully used in a variety of applications, such as archaelogy and city planning. Some examples of the former include a reconstruction of ancient Rome [89], Pompeii [123] and Mayan buildings [124], while prime examples of the latter are the Marseille urban planning project [43] and the Swiss Village in Masdar City [44], see Fig. 2.7.

2.1.5 Other Approaches

In this thesis (most notably in Part 1) we use the CGA shape grammar as our principal procedural modeling tool. Additionally, the grammars introduced in Part 2 and Part 3 are compatible with CGA and are converted into the appropriate format when necessary. Our decision is motivated by the proven usefulness of CGA in the field of urban modeling, human-readable syntax and the availability of a procedural engine to instantiate the models (CityEngine).

However, the methods presented in this thesis may be adapted to work with other types of procedural grammars and engines. One example would be the Centrale Procedural Architect (CPA) [161], which is many ways similar to CGA, but uses a different derivation scheme (DFS instead of BFS) and contains additional features such as consistency and differentiation tags.

The G^2 grammar [99] can be used to increase the degrees of freedom of simple boxes in CGA, allowing realistic creation of interconnected structures such as bridges or rollercoasters [98]. Finally, for building mass-modeling, one could also use a Manhattan-world rewriting system introduced by Vanegas et al. [195].

```
attr groundfloor_height = 4
attr floor_height = 3.5
attr tile_width = 3
attr height = 11
attr wallColor = "#fefefe"
window_asset = "facades/window.obj"

Lot --> extrude(height) Building

Building --> comp(f) { front : FrontFacade | side : SideFacade | top: Roof }

FrontFacade -->  split(y) {
  groundfloor_height : Groundfloor | { ~floor_height: Floor }* }

SideFacade -->  split(y) {
  groundfloor_height: Floor | { ~floor_height: Floor }* }

Floor --> split(x) {
  1 : Wall | { ~tile_width : Tile }* | 1 : Wall }

Groundfloor --> split(x) {
  1 : Wall | { ~tile_width : Tile }* | ~tile_width : EntranceTile | 1 : Wall }

Tile --> split(x) {
  ~1 : Wall | 2 : split(y){ 1: Wall | 1.5: Window | ~1: Wall }| ~1 : Wall }

EntranceTile --> split(x) {
  ~1 : SolidWall | 2  : split(y){ 2.5: Door | ~2: SolidWall } | ~1 : SolidWall }

Window --> s('1,'1,0.4) t(0,0,-0.25) i(window_asset)
Door --> s('1,'1,0.1) t(0,0,-0.5) i("builtin:cube")
Wall --> color(wallColor)
SolidWall --> color(wallColor) s('1,'1,0.4) t(0,0,-0.4) i("builtin:cube:notex")

```

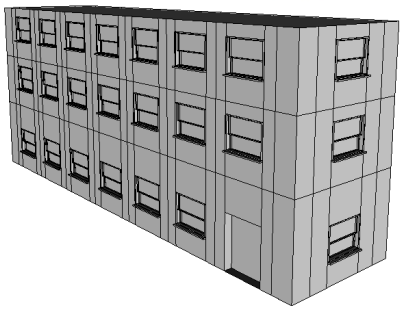


Figure 2.6: A simple building generated in CityEngine with its corresponding CGA code.

Figure 2.7: Use case scenarios for CityEngine [42]. Left: Marseille urban planning, right: Swiss Village, Masdar City. Image courtesy: Esri, Inc.

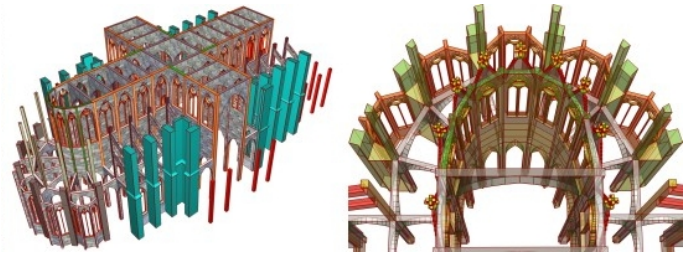


Figure 2.8: A basic model of the Cologne Cathedral, created using Generative Modeling Language. The whole model fits in 126 KB of GML code.

Finally, we mention Generative Modeling Language [76], or GML for short[1], a procedural modeling technique that allows algorithmical creation of complex 3D graphics. It is a stack-based programming language, inspired with Adobe PostScript, allowing the creation of high-level shape operators by combining low-level operators. GML can be used to create quite complex models such as Gothic Cathedrals in a few kilobytes of code, see Fig. 2.8. Although quite powerful, allowing procedural modeling of subdivision surfaces and free-form meshes, the syntax of GML is arguably more complex than CGA and perhaps better suited for automatic generation by specialized tools (similar to PostScript drivers).

[1]Not to be confused with the previously introduced CityGML, where GML stands for Geography Markup Language

Part I

Grammar-Based Reconstruction

Chapter 3

Architectural Style Classification

In recent years, procedural modeling has proven to be a very valuable tool for automatic reconstruction of architectural scenes. However, current algorithms rely on the assumption that the building style is known in order to load the appropriate procedural grammar. Determining the building style has been either implicit, or performed as a manual task. In this chapter[1] , we propose an algorithm which automates this process through classification of architectural styles from street-side images. Our classifier first winnows the set of input images so that only images containing buildings remain. Afterwards, the system separates individual facades within a single image and determines the building style for each building. This information can then be used to initialize a procedural reconstruction process, by loading the appropriate style grammar. We have trained our classifier to distinguish between several distinct architectural styles, namely Flemish Renaissance, Haussmannian and Neoclassical. We demonstrate the results of our approach on various street-side images collected.

[1]This chapter is based on the joint work with Markus Mathias, Julien Weissenberg and Luc Van Gool, published in 3D-ARCH 2011 [116]. While all parts of this work are a result of joint work and discussion, the first author Markus Mathias contributed in facade rectification and NBNN classification. Anđelo Martinović was mainly involved in scene classification and facade splitting.

3.1 Introduction

It takes several man-years to accurately model an existing city such as New York or Paris (e.g. 15 man years for the New York model in the King Kong movie). As said earlier, procedural modeling provides us with a faster alternative, by reconstructing a detailed model of a building from a set of images, or even from a single image. Considering the vast diversity of buildings and their appearances in images, the underlying optimization problem easily becomes intractable if the search space of all possible building styles has to be explored. Thus, all currently available inverse procedural modeling algorithms narrow down their search by implicitly assuming an architectural style. Müller et al. [126] developed a method based on the identification of repetitive patterns of regular, grid-like facades. Teboul et al. [182] demonstrate their procedural modeling approach based the Haussmannian architectural style, ubiquitous in Paris. The system of Vanegas et al. [195] assumes that building outlines follow a Manhattan-world grammar. In all cases, the style grammar is considered a given. Whereas for landmarks this may be derived from Wikipedia page coming with their images [140], street-side imagery typically does not come with style information.

We tackle the problem of grammar selection by proposing a four-stage method for automatic building classification based on the architectural style. The style information can then be used to select the appropriate procedural grammar for the task of building reconstruction. In this chapter, we demonstrate our approach on three distinct architectural styles: Flemish Renaissance, Haussmannian, and Neoclassical. Please note that we use a loose interpretation of these architectural terms, as our focus is on the categorization of building appearance, not actual provenance. For example, our Flemish Renaissance dataset also contains buildings from the Flemish Renaissance Revival style, which both have similar visual features. In addition, we created a publicly available dataset of facade images spanning the three presented styles, taken from the cities of Leuven, Antwerp and Brussels, in Belgium.

Interestingly, little research has been carried out in the field of architectural style identification. Romer and Plumer [146] aim at classifying buildings belonging to Wilhelminian style from a simplified 3D city model. However, their approach is based on a few coarse features (building footprint and height), with no image support.

Available image classification systems such as [18] often distinguish between images whose appearances are very different. Much focus has been on distinguishing indoor from outdoor scenes [136, 174]. Conversely, facade pictures share many common features, regardless of their styles. For instance, simple

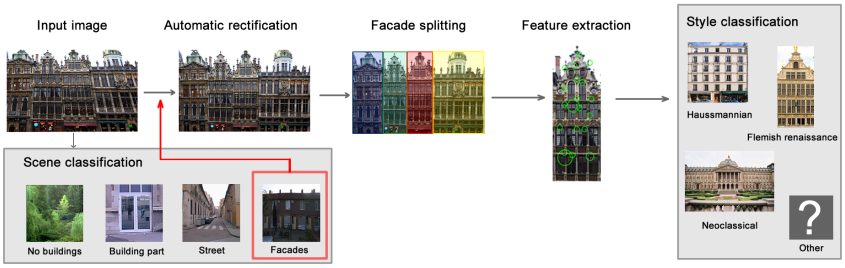


Figure 3.1: Overview of our architectural style classification approach.

colour or edge cues are insufficient to classify Haussmannian vs. Neoclassical buildings.

To the best of our knowledge, we are the first to tackle the problem of image-based architectural style identification. Our system provides a systematic and comprehensive way of estimating the building style from a single street-side image, incorporating steps of scene classification, image rectification, facade splitting and style classification.

3.2 System overview

The overarching goal of this work is to automate the process of modeling cities from images. Thus, we must take into account the problems arising from the setup of cameras mounted on a mobile mapping van. In this setup, cameras typically capture street-side images every few seconds. It is likely that a significant number of captured images will not even contain buildings, but other integral parts of urban areas, such as parks or trees. In very narrow streets, only small parts of buildings could be captured, insufficient for style classification. We therefore consider the additional problem of filtering out only those images that are useful for building modeling.

Fig. 3.1 gives an overview of our proposed approach. The first step is to determine if the image actually contains building facades (Sec. 3.3). If this condition is met, we attempt to rectify the image (Sec. 3.4), as the images of buildings taken from street level can contain significant perspective distortions. After the image has been rectified, we still face the problem of identifying individual buildings in the image. Urban areas can contain long, unbroken building blocks, but the architectural styles may vary from facade to facade. In Sec. 3.5 we use edge features in a heuristic approach to find separators of

individual buildings. Finally, we extract SIFT and self-similarity features from each facade, and use a Naive-Bayes Nearest-Neighbor (NBNN) classifier to determine the architectural style of the facade (Sec. 3.6). The obtained results are summarized in Sec. 3.7.

3.3 Scene classification

Mobile mapping images come with different content and quality. There are typically several cameras mounted on a van, with different viewing directions. Therefore, the first step in the process of building classification consists of winnowing all the collected images into a set of images that actually contain objects of interest. We want this step to be as fast as possible, due to the fact that it will have to deal with all images taken. On the other hand, the algorithm is desired to have good generalization to robustly deal with novel scenes. It has been shown by Oliva and Torralba [134] that humans have the capability to determine the type of the scene in less than 200ms. This abstract representation of the scene is called *gist*, and has served as a starting point for the development of numerous algorithms for fast scene classification [159, 133]. These holistic algorithms attempt to capture the global scene properties through various low-level image features. The suitability of different gist-based approaches for scene categorization is discussed by Siagian and Itti [160]. Due to its simplicity, speed and suitability for global scene classification, we opt for a gist-based scene classification step.

3.3.1 Scene classes

Based on the inspection of our collected dataset, we identify four most common scene types in street-side imagery (see Fig. 3.1)

- No buildings - images not containing any buildings. Typical examples in urban scenarios are parks, gardens and waterfronts.
- Street - images containing facades captured at a high angle to the facade planes, occurring when the camera orientation coincides with street direction.
- Facades - images containing one or more facades in their entirety.
- Building part - images containing only a small part of a facade, not enough for a complete classification or reconstruction.

Buildings	None	Part	Street	Facades
None	100	0	0	0
Part	2.8	85.6	2.4	9.2
Street	0.8	1.2	98	0
Facades	0	7.2	0.4	92.4

Table 3.1: Confusion matrix for the scene classification algorithm. All values are in percent.

Among the listed scene classes, only the "facades" class enables us to attempt a complete facade reconstruction. In an image produced by a sideways-mounted camera, the appearance of the "No building" class in collected images gives us the information about a gap in the building block, and that no buildings should be reconstructed. Similarly, if an image is classified as "Street", we can deduce the existence of a street crossing. Finally, the "building part" class informs us that the building is too large (or the street too narrow) to be captured in a single image.

3.3.2 Feature extraction and classification

In our implementation of scene classification, we use a similar approach to Torralba *et al.* [188]. We use a steerable pyramid of Gabor filters, tuned to 4 scales and 8 orientations. Filter outputs are then averaged on the 4×4 grid. This produces a feature vector comprising of 512 features. Classification is performed using a Support Vector Machine (SVM) with a Gaussian radial basis kernel function. The SVM is trained using a one-versus-all approach.

The scene classification dataset contains 1616 images in total, split into 4 classes of 404 images. Apart from using our own images from Leuven and Antwerp, we extracted additional images from publicly available datasets [151, 187, 178]. The images were then resized to a common size of 256×256 pixels and sorted into appropriate classes. Training and test sets are extracted randomly from the complete dataset, by taking 354 images of each class for training and 50 for testing.

3.3.3 Results

The process of training and validation is repeated five times with different splits into training and test sets, to eliminate possible biases in the choice of the

training set. Results were then averaged, resulting in the confusion matrix in Table 3.1.

We can see that the most distinct classes are easily separated from the others. Utilizing the vocabulary from Oliva and Torralba [133], we can deduce that the images from the 'No building' class usually have a high degree of *naturalness*, while the 'Street' class, characterized by long vanishing lines, has a high degree of *expansion*. The misclassification mostly occurs between classes 'Building part' and 'Facades'. This behavior is expected, because the scenes are visually quite similar.

3.4 Image rectification

Facade images are often taken in narrow streets, where purely sideways-looking cameras have a low chance of capturing the complete facade, as opposed to cameras looking obliquely forward, upward or backward. The facades in images taken by the latter type of cameras are projectively distorted. Prior rectification of images to a fronto-parallel view is a prerequisite not only for our facade splitting algorithm but also in further processing steps. In our implementation we followed the approach from Liebowitz and Zisserman [109]. After the scene classification step from Sec. 3.3 we assume that the image contains a planar surface containing two dominant perpendicular directions, which is a sensible assumption for man-made scenes.

The relation between points of the image plane $\mathbf{x}$ and points in the world plane $\mathbf{x}'$ can be expressed by the projective transformation matrix $\mathbf{H}$ as $\mathbf{x}' = \mathbf{H}\mathbf{x}$, where $\mathbf{x}$ and $\mathbf{x}'$ are homogeneous 3-vectors. The rectification follows a step-wise process (see Fig. 3.2) by estimating the parameters of the projective $\mathbf{P}$, affine $\mathbf{A}$ and similarity $\mathbf{S}$ part of the transformation $\mathbf{H}$, which can be (uniquely) decomposed into:

$$\mathbf{H} = \mathbf{SAP} \tag{3.1}$$

The projective transformation matrix has the form

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{pmatrix} , \tag{3.2}$$

where $\mathbf{l}_\infty = (l_1, l_2, l_3)^T$ denotes the vanishing line of the plane. Parallel lines in the world plane intersect at vanishing points in the distorted image. All

Figure 3.2: Rectification process: (a) input image with dominant lines, (b) projective distortion removal (c) affine distortion removal (d) similarity transformation

vanishing points lie on l_∞. To find these vanishing points we detect lines in the image using the publicly available implementation of the state-of-the-art line detector [8]. Then we use RANSAC [49] to detect the two vanishing points in the image.

The affine transformation:

$$\mathbf{A} = \begin{pmatrix} \frac{1}{\beta} & -\frac{\alpha}{\beta} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

has two degrees of freedom represented by the parameters α and β. The knowledge of the perpendicular intersection of the dominant lines l_a and l_b is the only constraint we can impose, as we have no further knowledge about other angles or length ratios in the image. Therefore the affine part of the rectification process can only restore angles but not length ratios. As shown in [109], α and β lie on the circle with center

$$(c_\alpha, c_\beta) = \left(\frac{a+b}{2}, 0\right) \text{ and radius } r = |(a-b)| \tag{3.4}$$

where $a = -l_{a2}/l_{a1}$ and $b = -l_{b2}/l_{b1}$. If the image did not contain any affine distortions, the parameters would have the value $(0,1)^T$, so we choose the closest point on the circle to that point for the correction.

Finally the image gets rotated by the rotation matrix $\mathbf{R}$ to align the dominant lines with the axes, scaled by $\mathbf{s}$ and translated by $\mathbf{t}$:

$$\mathbf{A} = \begin{pmatrix} \mathbf{sR} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \tag{3.5}$$

3.5 Facade splitting

Urban environments often consist of continuous building blocks with little or no space between individual buildings. Additionally, each building in the block may have a different architectural style. Therefore, the style recognition system needs to be able to separate different facades in order to properly classify them. As man-made structures are usually characterized by strong horizontal and vertical lines, we choose to exploit them as the main cue for building separation. We assume that each pair of neighboring facades are separable with a vertical line. Similarly to Xiao *et al.* [207] we use the following heuristics:

1. Horizontal line segments on the building usually span only one facade.
2. Vertical lines which intersect a large number of horizontal line segments have less chance of being a valid facade separator.

3.5.1 Line segment detection and grouping

After the rectification step, the vertical lines in the image coincide with the direction of gravity. First, we use a line segment detector [72] to find salient edges in the image. The obtained line segments are grouped in three clusters. The first cluster contains horizontal line segments (with a tolerance of ± 10 degrees in orientation). Similarly, the second contains vertical line segments, while the third contains all other detected line segments. The last cluster will typically have a smaller number of elements, due to the predominance of two perpendicular orientations in urban scenery.

3.5.2 Vertical line sweeping

Next, we sweep a vertical line over the image. At each position of the line, we calculate two values: *support* and *penalty*.

Support is based on the number of vertical line segments in the proximity of the sweeping line. Every vertical line segment contributes a weight proportional to its length: longer vertical line segments provide more support for the sweeping

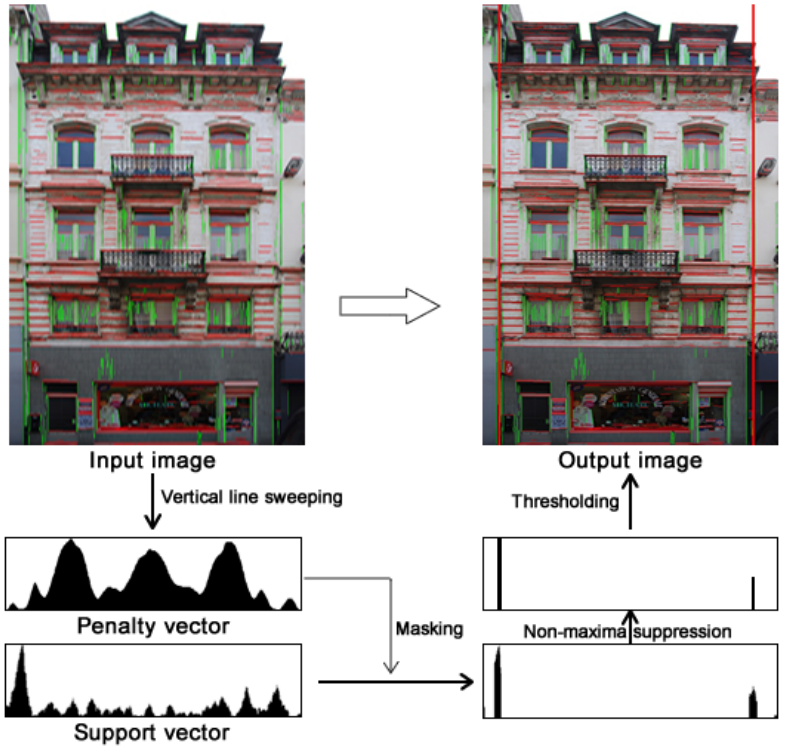


Figure 3.3: Facade splitting heuristic.

line. The support from neighboring line segments is reduced linearly with the distance to the sweeping line.

Penalty is calculated through the number of horizontal lines that the sweeping line crosses. Every horizontal line segment is weighted with its length: the longer the crossed segment is, the more penalty it generates. Relative position of the crossing point to the center of the horizontal line segment is also evaluated. Sweeping lines that cross horizontal segments near the edges will receive less penalty than those who cut the segments through the middle.

The result of the line sweeping process are two vectors of the same size, equal to the image width: support vector and penalty vector. We want to find the positions of the vertical line which correspond to local minima in the penalty vector and local maxima in the support vector. In order to calculate this, we first use the penalty vector to threshold the support vector. All of the line

positions which have more than 3% of the maximum penalty value are discarded. Then, positions which have less then 20% of the maximum support value are eliminated as well. We set the appropriate values in the support vector to zero. Finally, we perform local non-maxima suppression on the support vector through the use of a sliding window (9% of the image width). The resulting local maxima then coincide with the desired separator positions. We use these values to cut the building block into individual facades. The process of estimating facade separators from line segments is illustrated in Fig. 3.3.

3.5.3 Results

We test our facade splitting algorithm on a dataset consisting of 178 facade images from Brussels. We achieve a recall of 77% of the facade separators, with 29.4% false discovery rate. The cases where system failed to detect a boundary between facades were generally buildings which had strong horizontal features on the splitting lines. Highly protruding eaves from the neighboring roofs and shops with awnings which span multiple facades are typical examples. False positives generally appear on background buildings and non-planar facades.

3.6 Style classification

The style classification is an important step in order to select an appropriate grammar for the given building. To differentiate between the different styles, namely "Flemish renaissance", "Haussmann", "Neoclassical" and "Unknown", we got convincing results using the Naive-Bayes Nearest-Neighbor (NBNN) classifier proposed by Boiman *et al.* [16]. Despite its simplicity, it has many advantages. This non-parametric classifier does not need time consuming offline learning and it can handle a huge amount of classes by design. This means that new styles can easily be added. Furthermore it avoids overfitting, which is a serious issue for learning-based approaches.

3.6.1 NBNN algorithm

Algorithm 1 NBNN Image Classification

1: Compute descriptors $d_1, \ldots, d_n$ of the query image Q.
2: $\forall d_i \forall C$ compute NN of d_i in C: $\mathrm{NN}_C(d_i)$.
3: $\hat{C} = \arg\min_C \sum_{i=1}^{n} ||d_i - \mathrm{NN}_C(d_i)||^2$.

Style	Haussman	Neoclassical	Renaissance	Unknown
Haussman	98	0	0	2
Neoclassical	2	76	0	22
Renaissance	0	0	59	41
Unknown	3	0.5	0.5	96

Table 3.2: Confusion matrix for the architectural style classification approach, using SIFT features. The value in row i and column j represents the percentage of cases when class i was labeled as class j.

First we calculate SIFT [112] and SSIM [152] descriptors for our training images and sort them into the different classes. Then, for every descriptor d_i of the query image the nearest neighbor distances to each class are approximated using the FLANN library [122]. The sum over the Euclidean distances of each query descriptor d_i denotes the image-to-class distance. The class with the smallest distance is chosen as the winner class $\hat{C}$. The NBNN image classification approach [16] is summarized in Algorithm 1.

3.6.2 Results

Our dataset contains 949 images: 318 background facades (i.e. facades belonging to none of the trained styles), 286 images for Neoclassical, 180 for Haussmann and 165 for Flemish Renaissance. We have taken these images ourselves, except for the Haussmannian style images that come from the Ecole Centrale Paris Facades Database [178]. Table 3.2 shows the confusion matrix after cross-validation for the SIFT descriptor which was performing best throughout our experiments. While the Haussmannian style is clearly separated from other classes, many buildings of the Renaissance type are classified as "Unknown". While we have the least number of images for the Renaissance style, our definition for the class is very loose, resulting in a great diversity of the facades of that class. The mean accuracy for the SIFT features was 84%, while for the self similarity descriptor (SSIM) it reached only 78%. Fig. 3.4 shows the regions of the SIFT interest points labeled with different colors. Each color indicates the style to which the given feature had the minimum distance The features that respond to each style mostly appear on architectural elements typical for that style, e.g. features that appear on the capitals of columns in a neoclassical building.

Figure 3.4: Style classification: a) Neoclassical style (distinguishing features in red), b) Haussmannian style (features in blue), c) Renaissance style (features in purple) and d) Unknown style (features in green)

3.7 Conclusion

In this chapter we presented a system for automatic architectural style recognition. The output from this system can directly be used to initialize an inverse procedural modeling reconstruction.

In case the system doesn't recognize the style, the inverse procedural modeling system can try several possibilities or use default values. However, it comes with the cost of a more complicated optimization problem.

Furthermore, knowing the style of a building implies we know the kind of elements to look for during the reconstruction and their typical appearances. Moreover, the procedural reconstruction is then able to select accordingly the corresponding typical 3D models (or even textures) of the architectural elements to be used for rendering.

A possible extension of the proposed approach is to include feedback from a procedural modeling system, in order to perform online learning. For example, if the modeling is successful, the style database can be augmented with the current building. If not, we can select another style and retry the reconstruction.

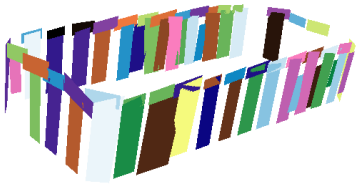
Chapter 4

Procedural 3D Building Reconstruction using Shape Grammars and Detectors

In this chapter[1] , we propose a novel grammar-driven approach for reconstruction of buildings and landmarks. Our proposed approach complements Structure-from-Motion and image-based analysis with a 'reverse' procedural modeling strategy. We reconstruct complete buildings as procedural models using shape grammars. In this chapter, we will assume that the procedural grammar is known; if this is not the case, we can run the approach from the previous chapter to select the appropriate grammar. The process can be seen as instantiating the grammar by determining the correct grammar parameters. As a case study, we have chosen the reconstruction of Greek Doric temples. This process significantly differs from single facade modeling due to the immediate need for 3D reconstruction.

[1]This chapter is based on the joint work with Markus Mathias, Julien Weissenberg and Luc Van Gool, published in 3DIMPVT 2011 [117]. While all parts of this work are a result of joint work and discussion, the first author Markus Mathias was mainly working with asset detectors, 3D reconstruction and the vision module. The main involvement of Anđelo Martinović was in the development of the grammar interpreter.

(a) Sparse point cloud



(b) Pruned detections in 3D

(c) One of the input images

(d) Our final reconstruction

Figure 4.1: Reconstruction of the Second Temple of Hera in Paestum, Italy

4.1 Introduction

Over the last years, the efficient creation of 3D models of single landmarks up to whole cities has received increasing interest, whereby landmarks play a particularly important role. Structure-from-Motion (SfM) approaches are popular methods to build such models from image sequences. They don't require expensive hardware and the images can be re-used for model texturing. Yet, SfM has problems which have proven to be difficult to solve [32]. It is doubtful whether further refinements to the typical SfM pipelines, i.e. better bottom-up processing, can overcome these issues. Therefore, it is worth trying to exploit prior knowledge about the scene.

We propose to create 3D models of buildings, by combining SfM, building element ('asset') detection, and inverse procedural modeling. The latter incorporates a shape grammar interpreter which drives the entire reconstruction process. The usage of asset detectors avoids fragile image or mesh segmentation processes and leverages recent progress in object class recognition. As grammars are

specific for a particular building style, like our Doric temples illustration, they need to be pre-selected correctly. As mentioned earlier, this is not critical, as one can use an automatic approach such as the one described in Chapter 3 or automatically mine images *and* information from Wikipedia pages of landmark buildings [140]. The Wikipedia pages typically specify the building style. This is also the case for the examples shown in this chapter. It is also important to note that the mined images often do not allow for a complete SfM reconstruction.

4.2 Related work

In the field of 3D architecture modeling, numerous approaches are available. In the following we give a short overview of representative works.

Cornelis *et al.* [32] present an approach for stereo-based, real-time, but simplified 3D scene modeling. Gallup *et al.* [58] has demonstrated a way to also handle non-planar surfaces. An approach for automatic partitioning of buildings into individual facades has been described by Zhao *et al.* [216], while Xiao *et al.* [207] use the concept of facades to reconstruct street-side models of buildings.

Probabilistic approaches have gained ground since the influential work of Dick *et al.* [39]. They use a model-based, Bayesian approach with Markov Chain Monte Carlo (MCMC) optimization. Alegre and Dellaert [3] use a stochastic context-free grammar and MCMC methods to deduce semantic information from building facades, relying on color constancy and rectangular shapes of facade elements. Ripperda and Brenner [145] use reversible jump MCMC methods for facade reconstruction, together with a formal grammar for facade description. Teboul *et al.* [182] use a coarse probabilistic interpretation of the facade to match the instantiated grammar model to the observed image. In order to find the grammar parameters, they kick-start the process with a pixel-wise segmentation and labeling step and then employ an algorithm for random-walk exploration of the grammar space.

We propose an approach that combines the robustness of a top-down grammar-based approach with the flexibility of the bottom-up image-based approach. Our main contributions are: (1) The reconstruction process is guided by the grammar. Instead of the developers having style-specific guidelines in mind when producing the system, a grammar interpreter tool renders the process more generic. It is the grammar that decides on what to do when. Moreover, structures that may not even be visible can be filled in. (2) Rather than relying on fragile segmentation processes to kick-start the semantic analysis, the grammar chooses the matching available detectors to assign initial semantic labels to image regions. (3) The system learns from its previous results. For instance, asset

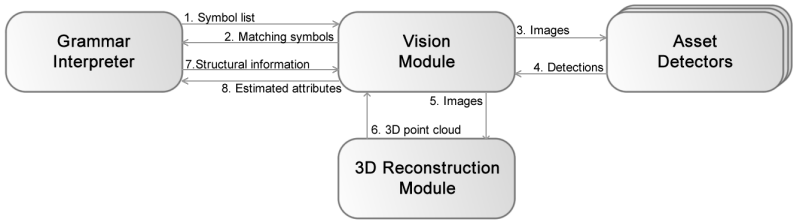


Figure 4.2: The proposed grammar-based reconstruction system.

detectors self-improve by using earlier results as additional training material. This also allows us to start with rather generic asset detectors, which have not been developed uniquely for the targeted style.

4.3 Main system components

Our grammar-based reconstruction system is composed of four main components:

- **Grammar interpreter:** Analyzes the input shape grammar, extracts semantic information and leads the reconstruction process.
- **3D reconstruction module:** Generates a 3D point cloud from the input images through uncalibrated SfM.
- **Asset detectors:** Extracts bounding boxes of 'assets' (building substructures) in the images.
- **Vision module:** Improves the detections by using 3D and semantic information from the grammar.

Note to reader: we use the term 'shape symbol' to refer to a string or name in the grammar, which refers to a class of shapes. In case a detector is available for that class of shapes, that type of shape is referred to as an 'asset'. Windows, doors, or pillar shafts are examples of assets in our system.

The system requires the following inputs: (1) a set of images of the building that is the target of reconstruction; (2) a database of asset detectors and (3) a style grammar which can express the target building. The above components are generic and have each been elaborated to the point where they support the Doric temple showcase. For instance, we have trained detectors of capitals

and pillar shafts, but would not have detectors for important elements in other styles yet (except for very general classes like windows and doors).

Fig. 4.2 shows how the parts of the system interact. First (1), the grammar interpreter initializes the vision module with a list of shape symbols automatically extracted from the grammar. They are then compared with the list of symbols that represent trained asset detectors from our database. The matching symbols (assets) are identified, reported to the grammar interpreter (2) and the detection (Sec. 4.3.2) process is initialized for those assets resulting in detection bounding boxes in all input images (3-4).

The image are also fed into the 3D reconstruction module ARC3D [196] to obtain a sparse 3D point cloud and the camera parameters from the building (5-6). For the matched symbols (detectable assets) the grammar interpreter parses the grammar to find structural information like spatial configuration or repetitions of these symbols (step 7).

The vision module (Sec. 4.3.4) uses a plane fitting algorithm to extract the dominant planes of the building. The detections from all images are projected into 3D and re-weighted based on consensus in 3D and the structural information. The output of this vision module are the sizes of the detected assets and their color, the footprint for the building and the parameters for the structural configurations (step 8). Then the building can be instantiated by the grammar interpreter by directly using the extracted parameters.

4.3.1 Grammar Interpreter

In this chapter we use the *CGA Shape* grammar for the description of our procedural models. Please refer to Sec. 2.1.4 for more details about the grammar definition.

Automatic Extraction of Semantic Information

In order to get the semantics of the building from a given shape grammar, we automatically construct a tree-like structure. Its nodes represent shapes, split, component split and repeat operations, capturing the structure of the building. The process begins with the extraction of shape symbols, and their classification as terminal or non-terminal shape symbols. In the next step, the rule set is analyzed, creating the tree structure. The interpreter also extracts the attributes from the grammar and assigns them to the appropriate nodes in the tree. An example grammar and its tree structure are shown in Fig. 4.3.

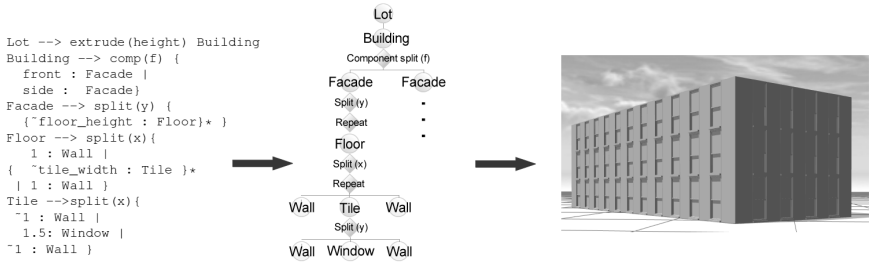


Figure 4.3: An example CGA grammar is shown on the left. The resulting shape tree is in the middle, and the rendered model with the default values on the right.

After the interpreter has analyzed the input grammar, the extracted symbols are fed to the vision module, which then returns the list of detectable assets. The assets constrain the grammar interpreter to extract only structure and composition information pertaining to assets. For each asset, it queries the semantic shape tree to retrieve the number and direction of repeat configurations the shape symbol appears in. This information is then sent to the vision module to re-weight the existing detections (see Sec. 4.3.4).

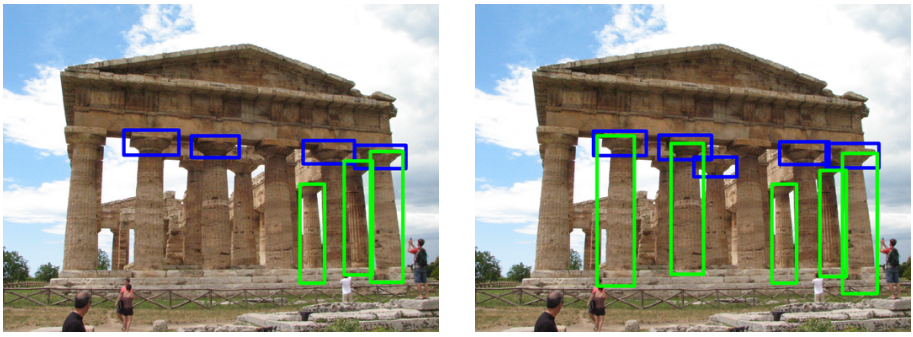
In the next step, we perform queries on all pairs of assets, determining their possible mutual composition. Assets typically correspond to multiple shapes in the shape hierarchy. Therefore, we check if all of the instances of one asset are in the same configuration with instances of a second asset. The configurations we can extract from the shape tree are:

- One shape is part of another shape
- One shape is on top/left/bottom/right of the other, relative to parent scope

For the Doric temple example, the system notices that capitals are on top of pillar shafts, and such coupling information is passed on to the vision module. Similarly, it would notice that windows are parts of facades, but not always on top of doors.

4.3.2 Asset detector

An important part of the strategy is to keep available a large set of asset detectors. We have used the Deformable Parts Model (DPM) by Felzenszwalb



(a) General detector (b) Specialized detector

Figure 4.4: Comparison of the general detector and the retrained specialized one.

et al. [46], trained on a few hundred hand-labeled examples for each asset. The detectors output bounding boxes of image regions where the asset was found, together with a score. Of course, there are the usual false positives and false negatives. The exploitation of the grammar helps the vision module in re-weighting or pruning those or just reducing their weight.

Another important aspect of our system is its ability to improve the detectors based on its previous 'experience'. For instance, the capital and shaft detectors that are activated to handle the Doric temples in this chapter have been trained on a diverse set of examples, including Ionic and Corinthian style in addition to Doric ones. As the system arrives at high confidence detections during the re-weighting process, it can then collect specific training examples to specialize the current general detector to one better suited for Doric temples as shown in the example Fig. 4.4. This online learning increases the chances of success to reconstruct the next Doric temple.

4.3.3 3D reconstruction module

For the creation of a 3D point cloud from the images of a building, we use the publicly available, online web service ARC3D [196]. This system employs a self-calibrating SfM approach, automatically estimating the camera positions and calibrations. The meshed surfaces provided by ARC3D are not used, as its 3D information is only needed to support our system and not to deliver complete parts of the model. One can imagine that one might eventually want

to use part of the ARC3D meshes for ornamental structures, if they were not available as assets.

4.3.4 Vision module

While the grammar interpreter guides the reconstruction process, the vision module gathers the information from the 3D data, the detectors and is responsible for substantiating the semantic information extracted from the grammar. It consists of four main components.

1. The **plane estimator** extracts the dominant planes from the sparse 3D point cloud.
2. The **3D reasoning module** is responsible for projecting the 2D object bounding boxes from the images into 3D and to estimate the assets sizes.
3. The **spatial configuration module** re-weighs the matching assets by using detected relations between different assets.
4. Eventually, detections for assets that appear in a repeat rule of the grammar are enhanced by **similarity detection**.

The modules for 3D reasoning, spatial configuration and similarity detection implement a re-weighting scheme (w^i_{3D}, w^i_{sp} and w^i_{sim}) of the detection score S^i_{det} of the i-th detection. The final score S^i_{final} is calcualted as:

$$S^i_{\text{final}} = S^i_{\text{det}} \cdot w^i_{\text{3D}} \cdot w^i_{\text{sp}} \cdot w^i_{\text{sim}} \tag{4.1}$$

Plane estimator

As the basic algorithm to extract facades from the point cloud we apply RANSAC [49]. To improve the quality of the detected planes we reduce the point cloud to points that project into detection bounding boxes in the images. This leads to planes intersecting the assets of interest. We set the inlier threshold proportional to the size of the point cloud. We stop extracting planes when the number of inliers of the final estimate is less than a given fraction of the total number of points. Furthermore, as soon as we have more than two planes detected, we calculate the gravity vector and the ground plane through the vector product of the plane normals, under the assumption of vertical planes. The footprint of the temple is extracted from the intersection of the ground plane with the facade planes. Fig. 4.5 illustrates the process of finding the planes.

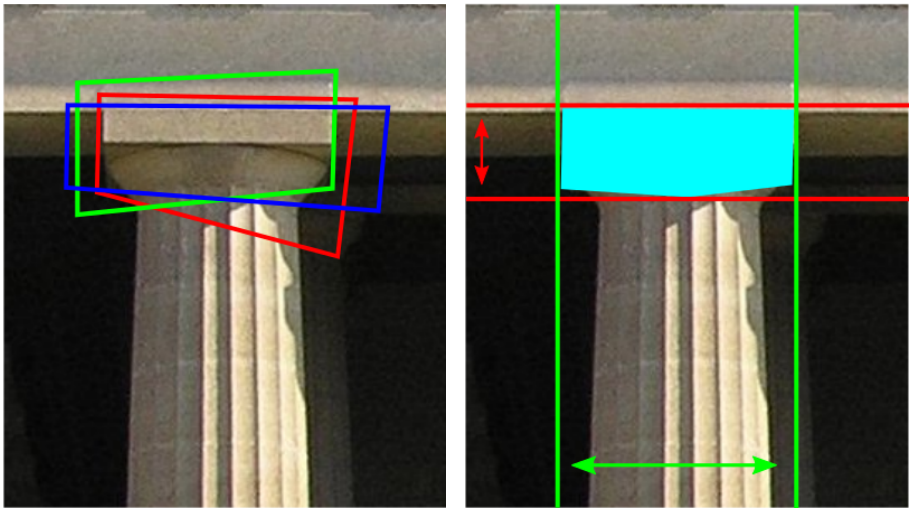
Figure 4.5: Plane estimation process: The first image shows the entire point cloud, then the planes are estimated in the reduced point cloud and shown in the cleaned complete point cloud.

3D reasoning module

The bounding boxes of detections from all views are back-projected to the planes detected in the sparse 3D model. Detections which overlap on the planes are then clustered. The clusters C_j are found in a greedy fashion. The detections are first sorted by their score. Starting from the best scoring detection as a cluster center, all overlapping detections are added to that cluster. A detection that does not overlap any previous bounding box defines a new cluster. The weight w_{3D} accounts for the size of the cluster and the 'rectangularity' of the detection. The latter is defined by the ratio $A_\mathrm{p}/A_\mathrm{br}$. The area of the polygon A_p is obtained by back-projecting an image detection bounding box into the 3D plane, while A_br is defined as the area of the corresponding polygon's minimum bounding rectangle. This rectangularity ratio decreases the influence of detections that come from cameras with an oblique angle to the plane, as these result in stretched polygons in 3D. Thus, every detection is assigned to a cluster C_j, represented by the cluster center (the detection with the highest score in the cluster). The size of the cluster $|C_\mathrm{j}|$ is down-weighted with the number of cameras n_seen in which the cluster centre is visible.

$$w^i_\mathrm{3D} = \frac{A_\mathrm{p}}{A_\mathrm{br}} \cdot \frac{|C_\mathrm{j}|}{n_\mathrm{seen}} \tag{4.2}$$

After thresholding by detection scores, cluster size and rectangularity, the remaining detections belonging to each cluster are used to find the spatial extent of the detected assets (see Fig. 4.6). The intersection area of these detections is orthogonally projected to the x and y axes (the x axis being parallel to the plane, and y axis being aligned with the gravity vector) to find the asset's dimensions (red and green arrow).



(a) Detections (b) Intersection and size estimation

Figure 4.6: Determining the assets size: the red and green arrows indicate the estimated height and width respectively.

Module for spatial configurations

This module uses semantic information originating from the grammar interpreter. The re-weighting w^i_{sp} is based on the spatial configurations of detections. The grammar interpreter informs the vision module about the possible spatial relations $(c_1 \dots c_k)$ between two elements. For a pair of elements a and b there are four relations possible, corresponding to the directions of split operations in the grammar: 'a left of b', 'a right of b', 'a above b', and 'a under b'. If one of these configurations is reported by the grammar interpreter, the vision module verifies them in the 3D plane.

$$w^i_{\text{sp}} = \frac{1}{k} \sum_{t=1}^{k} w^{i,t}_{\text{sp}} \tag{4.3}$$

$$w^{i,t}_{\text{sp}} = \begin{cases} \alpha & \text{if detection fits } c_t \\ 1 & \text{otherwise} \end{cases} \quad , t \in (1 \dots k)$$

Figure 4.7: Similarity voting space for a single detection (red rectangle).

The score of every detection that appears in a given relation is boosted by a constant multiplier α (in our experiments $\alpha = 1.1$).

Similarity detection

When the grammar interpreter informs the vision module about an asset appearing in a repeat rule, it expects as an answer the repeat distance. The similarity detection not only extracts this parameter but calculates a new weight w^i_{sim} for the detections of the repeat rule. For detections with no information about repetition w^i_{sim} is set to 1. The presence of a repeat rule implies directly that the asset included in that rule will appear several times along a given axis. This module searches for this periodicity, and boosts detection performance in three ways. First, assets that have not been detected so far can be inferred by similarity to a detected one. Second, as the repeat is defined along an axis, the locations of repeated assets are constrained to a line. Finding these similarity lines determines w^i_{sim}. Third, the parameter for the repeat distance is found as a byproduct of the repetition detection.

Fig. 4.7 shows the structures deemed similar for the detection marked with a red rectangle.

Similarity voting For every image we create a global voting space, i.e. a 2D accumulator. The similarity voting is based on local image features around detected interest points. Each interest point $\mathcal{F}_t$ is described with a triplet $(\mathbf{p}_t, s_t, \mathbf{d}_t)$, denoting its position, scale and the feature descriptor. The algorithm

iterates over all asset detections in the image and uses an ISM-like voting scheme [108] to find similar detections.

For a single detection, we consider the set of all interest points inside its bounding box. Each of these points is assigned a vote vector $\mathbf{v}_i$ from its position to the center of the box. Let $\mathcal{F}$ denote all interest points not covered by any detection. For each interest point $F_j \in \mathcal{F}$ we calculate a vote vector $\mathbf{v}_j$ by finding the nearest neighbor feature vector as follows:

$$F_k = \arg\min_{F_i \in \mathcal{F}}(||\mathbf{d}_i - \mathbf{d}_j||) \tag{4.4}$$

$$\mathbf{v}_j = \frac{s_j}{s_k} \cdot \mathbf{v}_k \tag{4.5}$$

Every interest point F_j casts a vote by adding a Gaussian kernel with standard deviation σ centered at $\mathbf{p}_j + \mathbf{v}_j$ to the accumulator. The process is repeated for all detections of one type in the image.

In our implementation, we use Hessian Affine interest points [121], and SIFT feature descriptors [112]. The value of σ is set based on the mean detection bounding box size.

Vanishing Line Extraction Similarity lines are found independently in each image. The best similarity lines in each image are found by fitting a line through the maxima of its voting space using RANSAC [49]. Maxima of the voting space that lie on a similarity line but do not correspond to any detection in the image are used to reinforce new detection hypotheses.

The number of similarity lines detected in an image depends on the total number of planes and the given grammar. In the grammar, a split operation immediately after a repeat operation is a cue to search for more similarity lines per plane. An example in Doric temples is the colonnade: a repetition of columns, each split into shaft and capital assets.

Re-weighting Factor Calculation The similarity lines found in all images are back-projected onto the planes of the model. The back-projection lines all vote in a Hough space to find the globally best similarity lines. Detections not corresponding to these lines are considered as outliers and are re-weighted based on their distance d to the lines.

$$w^i_{\text{sim}} = 2^{-(\frac{4d}{\Delta})^2} \tag{4.6}$$

The value for Δ is the mean detection bounding box extent along the axis perpendicular to the similarity line. For example, a detection centered exactly on the similarity line receives a weight of 1, while a detection offset by a quarter of mean bounding box size ($d = \Delta/4$) is weighted with 0.5.

Repeat Distance The estimated distance between maxima in the voting space is robust to small deviations of the detection position. Each detection that is not perfectly centered at the detected asset creates a voting space with the maxima shifted by the same amount from the ideal position. This results in a stable repeat distance. For a fronto-parallel view, an extra voting space is generated tracking these distances for all detections in all images. The maximum of that voting space is the parameter used as the repeat distance, or the repetition period. By using frontal views, we reduce the effect of errors in the plane detection process.

4.4 Grammar attribute estimation

At the end of the recognition stage, we have obtained the estimated values of asset sizes and the spacing of assets in repeat configurations. Additionally, we get the estimated size of the building footprint. The grammar interpreter then translates these parameters into the appropriate grammar attributes. In a typical scenario, the grammar will have additional attributes that cannot be estimated using the asset detectors alone (e.g. ornaments). For these attributes we use the default values present in the grammar. This approach enables us to "give an educated guess" for objects not visible in the images, but which have to be there due to the structural constraints imposed by the grammar.

4.5 Case Study - Doric Temples

Classical temples conform to strict architectural rules, which have been thoroughly analyzed in literature [171]. These rules have been converted into a CGA shape grammar representation. To demonstrate the usability of our method on real-world examples, we collected images from three different Greek Doric temples. The first is the Parthenon in Nashville, Tennessee, a full-scale replica of the original Parthenon in Athens. The remaining two reconstruction targets are the well-preserved remains of two temples in the ancient Greek city of Poseidonia (later renamed by Romans to Paestum), in today's province of Campania, Italy. The two temples we consider are the Temple of Athena (also

known as Temple of Ceres) and the Second Temple of Hera (also known as Temple of Neptune, or Temple of Poseidon). The reconstruction results are summarized in this section. Fig. 4.1 illustrates several steps of the reconstruction process for the Second Temple of Hera.

4.5.1 Asset Detectors

To train our asset detectors we use the publicly available implementation of Felzenszwalb's detector [46]. To cope with the higher variability in different types of capitals we have trained this detector as a two-component detector, whereas the shaft detector consists of only one component. For our first detector, we hand-labeled a few temple images of different styles, resulting in 189 annotations for capitals and 204 for shafts.

The reconstruction of the Second Temple of Hera resulted in 188 + 124 (capitals+shaft) newly gathered samples that we added to the training set, now specialized for Greek Doric temples. Keeping the false positive rate fixed at 2.2% for capitals and 5.4% for shafts we increased our detection rate by 7.31% and 14.89%, respectively.

4.5.2 Temple Grammar

A very simplified version of a grammar that describes classical temples is described in this section. We focus on the colonnade (the sequence of columns) as this is the most relevant part which contains our detectable assets.

```
Colonnade -->  split(x){Column | {columnSpacing:Column }* | Column}

Column    -->  split(y){shaftHeight:Shaft | capitalHeight:Capital}

Shaft     -->  i(shaftAsset)

Capital   -->  i(capitalAsset)
```

The colonnade is first split in the x direction into three parts: one column on each side, and a repetition of columns in the center. The side columns are handled separately resulting in a different spacing between the repeated columns and the the spacing to the left and right column. This fact is directly captured in the grammar rules, but cannot easily be inferred from the images alone. A column is further divided into capital and shaft. Due to this rule, the grammar interpreter informs the vision module about the relation "capital on top of shaft". The insert rule replaces the 3D volume by an asset from the database. As seen in the excerpt above, the parameters for column spacing, capital height

	Reconstruction	Original	Ratio
temple width	24.26	24.26	1.0
temple length	58.51	59.98	0.98
shaft width	2.13	2.11	1.01
column height	9.28	8.88	1.04
capital width	2.56*	2.72	0.94
capital height	1.37*	1.04	1.32
inter-column distance	4.45	4.48	0.99

Table 4.1: Size comparison for the Second Temple of Hera.

and shaft height appear directly in these rules. The remaining parameters, namely the column width, shaft width and temple color are extracted from the full derivation tree. The footprint size is estimated from the point cloud and not directly encoded in the grammar. Further parameters are either dependent on the estimated attributes or set to default values (e.g. the roof angle).

4.5.3 Results

Fig. 4.8 shows instantiations of the Parthenon replica in Nashville and the Temple of Athena, respectively. Properties such as the number of columns can easily be found from the detections. These are not grammar attributes but can be inferred through the instantiation process. In Table 4.1, we compare the dimensions of Second Temple of Hera with our estimations. All parameters are scaled to the size of the temple width. Sizes measured in the images are marked (*), while the groundtruth sizes are taken from the Perseus Digital Library [191]. Column height is the size of capital height + shaft height. The large error for the capital height can be explained by the fact that the pictures were taken from the ground. This results in the capitals appearing taller than they are in reality.

4.6 Conclusion

This chapter has introduced a novel way of 3D building reconstruction using shape grammars, where the grammar drives the reconstruction process. Additionally, object detectors are demonstrated to provide a good starting point for estimation of the grammar parameters. Furthermore, the system improves itself by automatically specializing the applied detectors. The validity of our approach is shown on a case study of classical Doric temples.

Figure 4.8: Procedural reconstructions of the Parthenon replica in Nashville (top row) and the Temple of Athena (bottom row).

Possible extensions of the presented approach include extending the supported set of CGA rules from which information can be extracted. Furthermore, a matching phase between the estimated model and the original images can be added to verify and fine-tune the estimations of the parameters. This matching can be used to include extra shapes via ARC3D meshes (like ornamentation) and to add effects of destruction and erosion as parts to be displaced or taken off the original model.

Part II

Grammar-Free Reconstruction

Chapter 5

A Three-Layered Approach to Facade Parsing

In this chapter[1] , we focus on the problem of determining the structure of building facades when a pre-defined procedural shape grammar is not available. We propose a novel approach for semantic segmentation which consists of three distinct layers, representing different levels of abstraction in facade images: segments, objects and architectural elements. In the first layer, the facade is segmented into regions, each of which is classified as one of the semantic labels, such as windows, doors, balconies etc. We evaluate different state-of-the-art segmentation and classification strategies to obtain the initial probabilistic semantic labeling. In the second layer, we investigate the performance of different object detectors and show the benefit of using such detectors to improve our initial labeling. The generic approaches of the first two layers are then specialized for the task of facade labeling in the third layer. There, we incorporate additional meta-knowledge in the form of *weak architectural principles*, which enforces architectural plausibility and consistency in the final reconstruction. Rigorous tests performed on two existing datasets of building facades demonstrate that the proposed system outperforms state of the art, even when using outputs from lower layers of the pipeline. Finally, we demonstrate how the output of the highest layer can be used to create a procedural building reconstruction.

[1]This chapter is based on the joint work with Markus Mathias and Luc Van Gool, submitted to IJCV. Anđelo Martinović and Markus Mathias share first authorship. Markus Mathias focused mainly on the object detectors and implementation of weak architectural principles. Anđelo Martinović worked on the semantic labeling in the first layer, CRF formulation and learning in the second layer, and GA optimization in the third layer.

Figure 5.1: The three-layered approach takes a cropped and rectified facade image as input (left). The result of processing is a labeled output image (middle), where each color corresponds to one semantic class. From this labeling we produce a textured procedural model (right).

5.1 Introduction

Accurate reconstruction of building facades plays an important role in 3D city modeling. Current models built by simple plane fitting and texturing are a good starting point, but provide inadequate 3D visual perception. For instance, artifacts in the 3D shape often show up during unrestricted user movement around the model. Due to diversity of appearance, hierarchical structure of scene objects and the lack of implementing long-range interactions, it appears impossible that improved, bottom-up depth extraction and primitive fitting alone can avoid such artifacts from sneaking in. Furthermore, conventional bottom-up models based on structure from motion lack any semantic knowledge about the scene. Yet, adding a good understanding of what needs to be modeled is a strong cue, not only to improve the visual and 3D quality of the model, but also to substantially widen its usage (e.g. for animation where people should walk through doors, not walls, when wanting to know the average number of floors that the buildings in a street have, etc.). Fig. 5.1 shows an example of our modeling pipeline, that builds on the inclusion of semantic aspects.

As mentioned at the beginning of the thesis, *procedural modeling* provides an effective way to create detailed and realistic 3D building models that do come with all the semantic labels required. Yet, the solutions that create procedural models for *existing* buildings from images or other data sources typically start from a preprocessed version of the raw data. The semantic segmentation of facades - also referred to as facade parsing - is a good example. This said, such

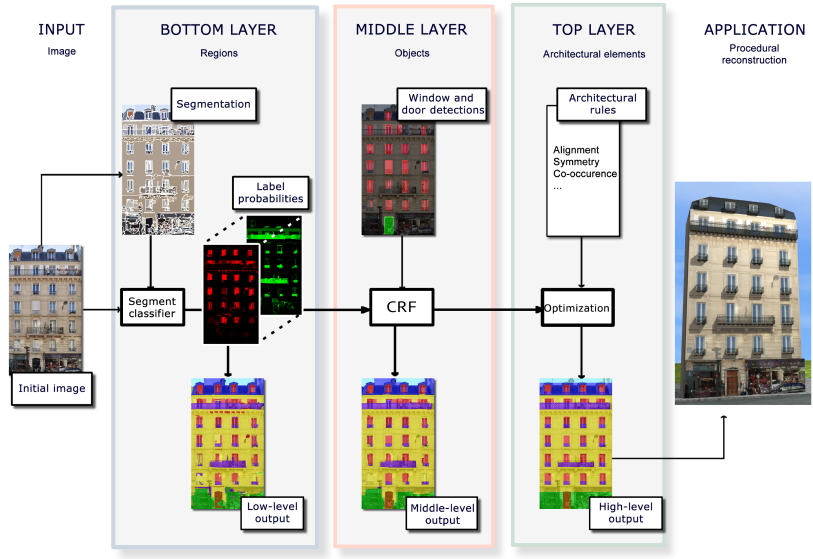


Figure 5.2: The proposed three-layered approach to facade parsing.

accurate labeling of facade elements (such as windows, doors or balconies) is a difficult problem in its own right, given the great diversity of buildings and the interference of factors like shadows, occlusions and reflections in the images. It is this facade parsing that this chapter focuses on. Furthermore, a shape grammar specific to the desired style is not easy to come by. An expert in that style needs to sit down with a person versed in the creation of the grammars. Therefore, our approach also avoids the need for such a style-specific grammar and uses generic architectural principles instead. This stands in contrast to most earlier inverse procedural modeling work (see e.g. [181]). Assuming that the input facades are of a certain architectural style helps to keep the dimensionality of the search space a bit smaller. In the case of Teboul *et al.* [181], this is e.g. the Haussmannian style, ubiquitous in central Paris. Strong prior knowledge about this style is imbued in the Haussmann-specific procedural facade grammar.

This chapter extends our previous work [113], which at the time achieved top results on the task of facade parsing, even without using any style-specific prior knowledge. Still, if style information is available, it can be incorporated into the system through the usage of extra "architectural principles". In contrast to full procedural grammars, these principles do not encode the entire facade structure and can be formulated explicitly by laymen. Moreover, we demonstrate how procedural rules and thus simple shape grammars can be derived from facade labeling, rather than vice-versa. By avoiding the need for a style prior, we

circumvent the manual construction of style-specific grammars.

Our approach to facade parsing is performed in three layers, representing different levels of abstraction in facade images: segments, objects and architectural elements. An overview is given in Fig. 5.2.

Bottom layer. Initially, the facade is segmented into superpixels, i.e. image regions. Visual features are extracted from the corresponding regions, and subsequently used for classification. Each region is assigned a probability distribution over semantic classes. In this layer, we pay particular attention to the evaluation of different segmentation algorithms and classifiers on the task of semantic segmentation of facades, as well as the effect of segmentation coarseness on the classification performance.

Middle layer. The second layer of our approach introduces detectors for objects found in urban scenes, such as windows and doors. The classifier output from the bottom layer is combined with the object detector responses (see Fig. 5.2) and results in our improved middle layer output. The combination of detections and the labeling from the bottom layer is achieved through a 2D conditional random field (CRF) defined over the image, which can be efficiently solved with graph cuts. We investigate the performance of different object detectors and show the benefit of using such detectors to improve our initial labeling.

Top layer. The generic approaches in the first two layers are complemented with considerations dedicated to the task of facade labeling. In the top layer, we incorporate additional meta-knowledge in the form of *weak architectural principles*. In contrast to shape grammar rules, these principles are easily observable in the images. For instance, the principle of vertical window alignment is often an implicit consequence of grammar rules, never made explicit in any of them. Also, we use these architectural concepts as guidelines, not as hard constraints. Therefore, we are also able to model irregular facades, as demonstrated on the eTRIMS dataset that contains different facade styles. The architectural principles are designed such that each principle either proposes new facade elements, re-arranges their position, or evaluates the current configuration of elements. Finally, we pose the search for the optimal facade labeling as a sampling-based approach. Although the overall pixel accuracy of the semantic segmentation is not greatly influenced by the top layer, we obtain image labelings that are visually more pleasing, with clearly defined object boundaries and structures. These in turn form a stronger basis for further processing, e.g. for deriving style-specific procedural grammars.

While the overall structure of our system is similar to that of Martinović *et al.* [113], each layer has been upgraded. In the bottom layer, instead of using a fixed combination of Mean-shift [31] segmentation and the Recursive Neural Network (RNN) classifier [164], we evaluate various segmentation and classification algorithms. In the middle layer, we learn a prior on element locations and calculate the probabilistic detector output in a more robust way. Furthermore, we learn the CRF parameters with structured SVMs [190]. In the top layer, we propose a coupled subsampling-and-optimization technique in a generic framework that allows for addition of new principles.

The main contributions of this chapter are as follows:

1. A new approach for facade parsing, combining low-level information from the semantic segmentation, middle-level detector information about objects in the facade, as well as top-level architectural knowledge;
2. A rigorous evaluation on two different datasets which shows that we outperform the state-of-the-art in facade parsing;
3. The concept of *weak architectural principles*, which introduce the high-level knowledge needed for ensuring architectural plausibility.

5.2 Related Work

This section concisely describes the relation between the proposed work and prior art. We have organized this overview into several main topics.

Scene parsing. There exists a significant body of work in this field. Some approaches attempt to estimate labels for each pixel in the image [158, 54]. Others depend on an initial segmentation of the image into super-pixels. Visual features are extracted from the corresponding patches or regions, and subsequently used for classification. In our work, we opt for the region-based approach in the first layer, as state-of-the-art results in semantic scene segmentation are achieved by similar approaches.

These approaches ensure labeling consistency by incorporating region context in various ways: estimating geometric labels [66, 185], using multiple over-segmentations [100], learning segmentation trees [164] or label transfer combined with a simple MRF [111, 185]. However, facade structures are difficult to analyse with solely region-based approaches, as the initial segmentation boundaries might not correspond to actual object boundaries in the image. Our work

puts more emphasis on the combination of the region-based approach with higher-level information, such as object detectors and architectural knowledge.

Combining semantic segmentation with object detectors. The effect of positive reinforcement between semantic segmentation and object detection approaches has been demonstrated in several works. Heitz and Koller [77] use image regions as context for improved object detection. This is an orthogonal approach to our work, as we use object detectors to improve the facade labeling. Joint reasoning about pixel-wise labeling and object detectors in a CRF framework was performed by Wojek and Schiele [201], while also capturing temporal consistency for video sequences. However, the complexity of their CRF requires slow approximate inference with loopy belief propagation. The work of Ladicky *et al.* [102], later extended by Floros *et al.* [51], disregards the temporal consistency, but in their CRF framework inference can be performed efficiently via graph cuts. The second layer of our approach is similar to [102], but with two key differences. Firstly, instead of using detector outputs as higher-order potentials, we decompose them into unary potentials, which are learned based on detector output on the training set. This enables us to solve a much simpler CRF optimization problem. Note that the problem of inferring pixel-level cues (or masks) from bounding boxes can also be tackled by using per-exemplar detectors as in [184] if the objects exhibit high variability in appearance. The second advantage of our approach is that we can efficiently learn the CRF parameters on the validation set based on the structured SVM approach [190]. As shown by Szummer *et al.* [173], CRF parameter learning using graph cuts is tractable, fast, and much more efficient than methods based on cross-validation, especially for larger parameter vectors.

Urban reconstruction. For an extensive overview of the field, we refer the reader to the survey of Musialski *et al.* [129]. Our main focus is the semantic segmentation of isolated and rectified facades. These can be obtained from more general street-side imagery by approaches [216, 199, 141], or by our proposed approach in Chapter 3. Furthermore, we demonstrate that even in cluttered scenes with occlusions such as vegetation or cars, our approach can semantically segment the facades.

Xiao *et al.* [206, 207] target realistic visualization with a low level of semantic encoding in the reconstruction. In their work, facades are represented with planes or simple developable surfaces. On the other hand, many approaches employ higher-order knowledge for building reconstruction. Probabilistic approaches to building reconstruction started with the work of Dick *et al.* [39], where a building is assumed to be a 'lego' set of parameterized primitives. The inference

is performed using a Reversible Jump Markov Chain Monte Carlo (rjMCMC) approach. However, an expert is needed to set the model parameters and prior probabilities. In contrast, the free parameters of our system are learned from validation data.

Certain approaches are based on priors on the facade layout. A grid-based layout is a common assumption [93, 154, 208, 75]. The work of Müller *et al.* [126] also assumes a certain degree of facade regularity, and fits procedural grammar rules to the detected subdivision of the facade. Unlike the aforementioned methods, our approach poses no grid constraints on the facade.

Grammar-based approaches are quite popular in the field [3, 145, 74]. They allow the generation of very clean models and labeling results, demonstrated by approaches where facade reconstruction is postulated as a problem of finding the correct parameters of a pre-specified shape grammar. For example, Koutsourakis *et al.* [95] fit a hierarchical tree grammar to a rectified facade image using an MRF formulation. Teboul *et al.* [182, 161] extend this work by utilizing bottom-up segmentation cues and a random walk exploration of grammar parameter space. An improved search algorithm based on reinforcement learning is presented in [180, 181]. Depth cues have also been used in the context of grammar-based parsing [162], by transforming the problem into a multiobjective optimization, solved with a genetic algorithm. As said earlier, the focus of this chapter is the case when a predefined grammar is not available or not applicable for the architectural style at hand.

Object detection has also been considered in grammar-based approaches. As a reminder to the reader, in Chapter 4, we have shown how 3D reconstructions of Greek Doric temples can be created using a specialized procedural grammar, 3D Structure-from-Motion (SfM) point clouds, and object detectors. Several other approaches use detector outputs to augment the bottom-up merit functions for grammar-based facade parsing. Ok et al. [132] use a simple approach where the merit of undetected classes is zeroed out in every detection. In a work similar to our first two layers, Riemenschneider *et al.* [144] combine a pixel-wise classifier with Hough forest detectors using a MRF framework. This labeling is then used to create an irregular grid which is labeled by using a predefined grammar. In contrast to this work, we utilize much stronger bottom-up classifiers and detectors, without restricting the final output to a grid.

The benefit of relying on shape grammars is that they strongly restrict the search space during parsing. Yet, the grammar may not be expressive enough to cover the variance in real world data. Furthermore, an expert is needed to write the grammars for the relevant styles. Human intervention is also required to pre-select the grammar appropriate for each specific building. The latter requirement can be mitigated by applying style classifiers that automatically

recognize the building style from low-level image features, as shown in Chapter 3. Still, using a style-specific grammar would imply that it needs to be available beforehand, which at least for the moment is a limiting issue. Therefore, in the earlier version of this work [113], we did not assume the existence of such a predefined grammar. Other authors have also recognized the limitation of relying on expert-written procedural grammars, e.g. [35], replacing them with weaker, or learned priors. In fact, our guiding principle is to derive procedural grammars based on automatically parsed facades, rather than vice-versa.

In summary, the current state-of-the-art in semantic facade parsing needs the prior specification of a style-specific grammar. The goal of this chapter is to show that an approach can be designed that performs facade parsing with high accuracy without needing such a grammar, allowing us to deal with a wider variety of buildings. Moreover, the traditional pipeline can be reversed: we let the image parsing control the grammar inference, rather than simply using a fixed grammar to control the process of image parsing. Selecting the appropriate images for grammar induction can be automated by using style classifiers (Chapter 3), which, as said, require far less human interaction than the prior construction of entire grammars.

5.3 Datasets Description

Our facade parsing approach is evaluated on two datasets, the "Ecole Centrale Paris Facades Database Benchmark 2011" [178] and the eTRIMS database [94]. The ECP database provides labels for multiple facade elements, while the eTRIMS dataset also contains non-building classes, such as vegetation. Since we are primarily interested in the accurate parsing of building facades, our main focus will be on the ECP database. We additionally validate our approach on eTRIMS and show that we outperform previous state-of-the-art results.

The ECP Database contains 104 annotated images of single rectified and cropped facades in the Haussmannian style. The dataset has 7 different labels $\Psi = \{$*window, wall, balcony, door, roof, sky, shop*$\}$. We use the new and more precise set of annotations provided in [113]. Our evaluations are performed with a 5-fold cross-validation on this dataset. For each fold, we use 60 images for training, 20 for validation, and 20 for testing.

The eTRIMS Database provides accurate pixel-wise annotations and contains 60 images. Unlike the ECP dataset, the images are not rectified and the facades uncropped. We use the automatic rectification algorithm of Liebowitz and Zisserman [109] as a preprocessing step. To allow for a fair comparison to previously reported results, we 'un-rectify' our output prior to

evaluation. The labels of this dataset $\Psi = \{$*building, car, door, pavement, road, sky, vegetation, window*$\}$ are quite different compared to the ECP dataset, as there are several non-building classes. As in [210], we evaluate our algorithm by performing a 5-fold cross-validation with random sub-sampling. However, instead of using 40 images for training, we use only 30, leaving 10 images as a validation set. 20 images are used for evaluation.

5.4 Bottom Layer: Initial Semantic Segmentation

The purpose of the bottom layer is to provide the initial classification of each pixel into one of the semantic classes. As a single pixel does not contain enough information for accurate classification, one must consider its context.

In a *patch-based* approach (e.g. the baseline of [182]) the context of a pixel is an image patch of certain size, centered on the pixel. Each pixel is then classified separately, based on the features extracted from the corresponding patch. The downside of this method is that the final result can be quite noisy, since neighboring pixels can be assigned to completely different classes.

Another approach is to use *regions* (super-pixels), i.e. to segment the image in coherent regions, which ideally share the same semantic label. Classification is then performed on the region level, which provides three main advantages over the patch-based approach. First, since all pixels within a region share the same class, the result is generally less noisy. Second, the dimensionality of the problem is significantly reduced as the number of regions in the image is typically two orders of magnitude lower than the number of pixels. Third, coherent regions can provide a stronger clue for a classifier e.g. by their specific shape. Yet, any errors in the segmentation step will propagate to the classification, since the final labeling is restricted to follow the super-pixel boundaries.

In our work, we opt for a region-based approach, as state-of-the-art results in semantic scene segmentation have been achieved by similar approaches [66, 184, 100]. Our experiments validate this choice, as we show in Sec. 5.7. The implementation of a region-based classification approach consists of three steps: segmenting the images into regions, extracting features from the regions, and using a classifier to obtain probabilistic estimates of classes, or labels, for each region. In this section we investigate how different segmentation algorithms and classifiers affect the speed and quality of facade labeling.

5.4.1 Image Segmentation

One of the most important choices in region-based segmentation is the number of regions created. We define the *maximum achievable accuracy (MAA)* as the accuracy (pixel-average or class-average) obtained by using an oracle classifier, which assigns each region the label of the pixel majority in the ground truth. Clearly, a pixel-based oracle classifier achieves the *MAA* of 100%, since every pixel is classified separately. By using region-based segmentation we introduce the constraint that all pixels in a single region share the same class. On the one hand, a more fine-grained segmentation tends to result in a higher *MAA*. On the other hand, classifiers tend to perform better on discriminative and therefore larger regions. Even though coarse-grained segmentation is better suited for classification purposes, this process introduces errors when semantically different regions merge together, which reduces the *MAA*. Intuitively, finding a good segmentation of the image is equivalent to discovering the optimal trade-off between region size and discrimination potential.

Over the years, a large number of image segmentation algorithms have been developed [31, 48, 1, 6, 193]. In this work, we chose to evaluate three dissimilar algorithms on the task of facade segmentation. The first, *Mean-shift* [31], is a popular algorithm that was demonstrated to perform well for facade parsing in our earlier work [113]. Second, we evaluate one of the fastest segmentation algorithms to date, *SEEDS* [193]. This method was shown to have competitive results while running in real-time. Finally, the third algorithm in our comparison is *gPb* [6], which sacrifices running time for an accurate calculation of the segmentation tree. In order to perform a fair comparison to the other algorithms, we consider only a single level in the gPb tree.

5.4.2 Feature Extraction

We use the same feature extraction algorithm in all of our experiments. Following the procedure described in [66], we extract appearance (color and texture), geometry, and location features for each region. This choice was motivated by the fact that the same features are used in several top-performing scene segmentation approaches [66, 100, 164]. Additionally, the publicly available implementation in form of the Stair Vision Library [67] enables us to quickly extract features from pre-segmented facade images. With default parameters, this results in feature vectors of size 225.

5.4.3 Classifiers

Given its feature vector, our goal is to assign to each region one of the semantic classes described in Sec. 5.3. For this task, we consider five different multinomial classifiers:

1. LOG: Multiclass logistic regression classifier [67]
2. LOG-CRF: An extension of LOG, which adds a Potts-model CRF with superpixels as nodes. [67]
3. MLP: Multilayer Perceptron [38]
4. SVM: Multiclass Support Vector Machine [25]
5. RNN: Recursive Neural Network [164]

In each of the above classifiers, its output for each region can be interpreted as a confidence score of assigning the region to a certain class. These scores can be transformed into a probability distribution using a softmax function.

For the first two methods, a boosted one-vs-all classifier is learned for each class using Adaboost. Then, the outputs of the classifiers are used as features for learning the multiclass logistic model (LOG) with a linear predictor function. The LOG-CRF model is obtained by adding a pairwise term between neighboring segments, which has a smoothing effect. For more details about the implementation, please consult [67]. The multilayer perceptron we use is a feed-forward artificial neural network with a single input, hidden and output layer. The number of neurons in the input layer is 225, equal to the number of features. The output layer contains as many neurons as there are semantic classes. Using a rule-of-thumb that states that the optimal size of the hidden layer is usually between the size of the input and the size of the output layers, we set the number of hidden neurons to 75. As the SVM classifier we use the publicly available one-vs-one multiclass SVM with a Gaussian kernel function [25]. The parameters C and γ are determined from the validation set. Finally, the RNN classifier was shown to perform well for the semantic segmentation of general scenes [164] and building facades [113]. In line with [113], we set the length of vectors in the semantic space to 50.

5.4.4 Analysis

By setting the average number of regions per image to a fixed value, we evaluate the interplay between different segmentations and classifiers. Second, we select

the best combination of segmentation algorithms and classifiers, and investigate how changing the number of segments affects the classification accuracy. For completeness, we calculate both pixel-wise and class-wise accuracies. The former is defined simply as the percentage of correctly classified pixels. The class-wise accuracy is defined as the unweighted average of all class accuracies (the latter being the % of pixels of a class that were correctly classified), which provides an insight into classification performance on smaller classes. All of the presented experiments are performed on the ECP dataset.

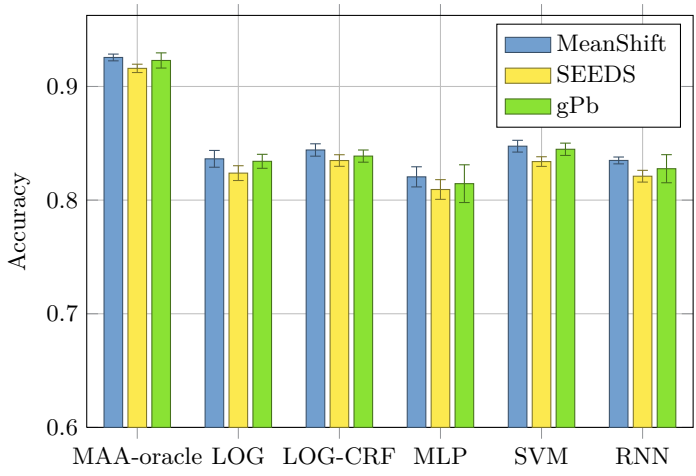
Segmentation and Classification

Keeping the average number of segments per image equal for all three segmentation algorithms (~ 690 segments), we evaluate the maximum achievable accuracy (MAA), as well as classification accuracy achieved with each of the classifiers from Sec. 5.4.3. The results obtained are shown in Fig. 5.3.

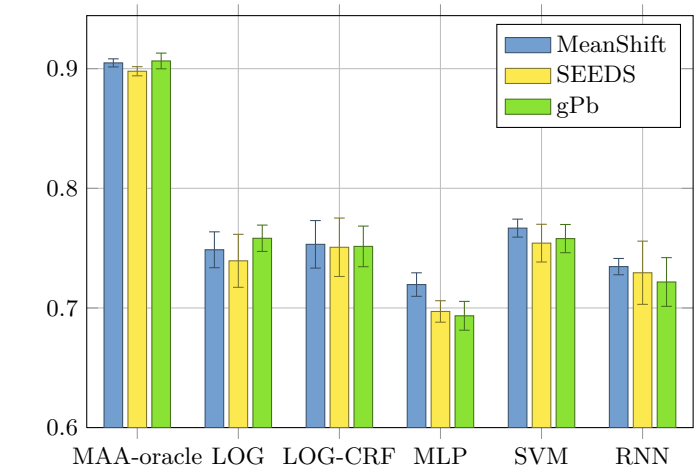
Generally, using SEEDS as the segmentation algorithm results in the lowest classification accuracy. However, the difference between SEEDS and its competitors is relatively small (around 1%), and one may opt to use SEEDS when speed is of the essence (as it may very well be when dealing with complete city modeling). Mean-shift and gPb performed similarly in each of the five classifier scenarios. Since Mean-shift segmentation is much faster to compute than gPb, we select it as our preferred segmentation algorithm.

Additionally, the data reveals that our method is quite robust with respect to the choice of classifier. As expected, there is a noticeable difference between the maximum achievable accuracy (MAA) and the results obtained with the five classifiers. The gap becomes even more apparent when considering class-wise accuracies. This is due to the unbalanced datasets, where the number of pixels of each class label varies significantly. By definition, class-wise accuracy accounts for this variation.

We can see that the LOG-CRF model benefits from the addition of pairwise terms, compared to the LOG classifier. RNN outperforms the basic MLP model, but the results do not justify the extremely long training time of RNN (around 24 hours). Unlike other methods, which classify each of the segments separately, RNN also creates a hierarchical parse tree of the image by recursively combining neighboring segments. However, existing RNN-based approaches [164, 113] do not exploit any knowledge from the tree during classification. Additionally, we achieved no improvement by using higher levels of the hierarchy, raising further questions about the usefulness of the tree. Finally, the SVM classifier emerges as the winner, as it achieves better results than its competitors both in terms of pixel-wise and class-wise accuracy.



(a)



(b)

Figure 5.3: (a) Pixel-wise and (b) class-wise accuracy of different segmentation algorithms and classifiers on the ECP dataset. The final results are calculated as the mean, and error bars as the standard deviation of results obtained from five cross-validation folds.

One may argue that although the SVM classifier has the best performance in the first layer, some other classifier might provide better bottom-up information to the other layers of the system. We tested this hypothesis with the LOG-CRF and RNN classifiers, but obtained no improvement over SVM.

Number of Segments

The results in the previous section were obtained by setting the average number of segments per image to a fixed value. Now we evaluate the effect of changing the segmentation coarseness while fixing the best performing segmentation - classification pair, i.e. Mean-shift and SVM. By changing the minimum region size parameter in the Mean-shift implementation, we obtain 7 different levels of coarseness, ranging from 1906 to 283 segments per image.

The classification results in Fig. 5.4 show that the maximum achievable accuracy steadily drops as we use coarser and coarser segmentations. The classifier performance follows a different trend, as its performance peaks around an optimal number of segments. While large segments introduce errors by combining neighboring objects into single regions, fine segmentations produce small image regions which are not discriminative enough for the classifier. However, this effect is prominent only when dealing with rather extreme numbers of segments, as we obtain similar results from 500 to 1000 segments per image. Therefore, we selected the middle level of coarseness in Fig. 5.4, amounting to 691 segments per image, on average.

5.5 Middle Layer: Introducing Objects through Detectors

In the middle layer, we enrich our labeling pipeline by localizing facade elements directly through the usage of object detectors. Such detectors search for coherent structures that can span several of the previously segmented regions, thus allowing better discrimination. In this section, we demonstrate how detectors are integrated into our system and argue that their usage benefits the overall labeling quality.

Our bottom layer provides a probability distribution over labels for each region (segment) in the image. These regions are determined using fixed segmentation parameters for all input images. As shown in Fig. 5.3, even with the perfect *MAA* oracle, we can reach at most 92% pixel accuracy and 90% class accuracy. By using object detectors in the second layer, we not only provide information

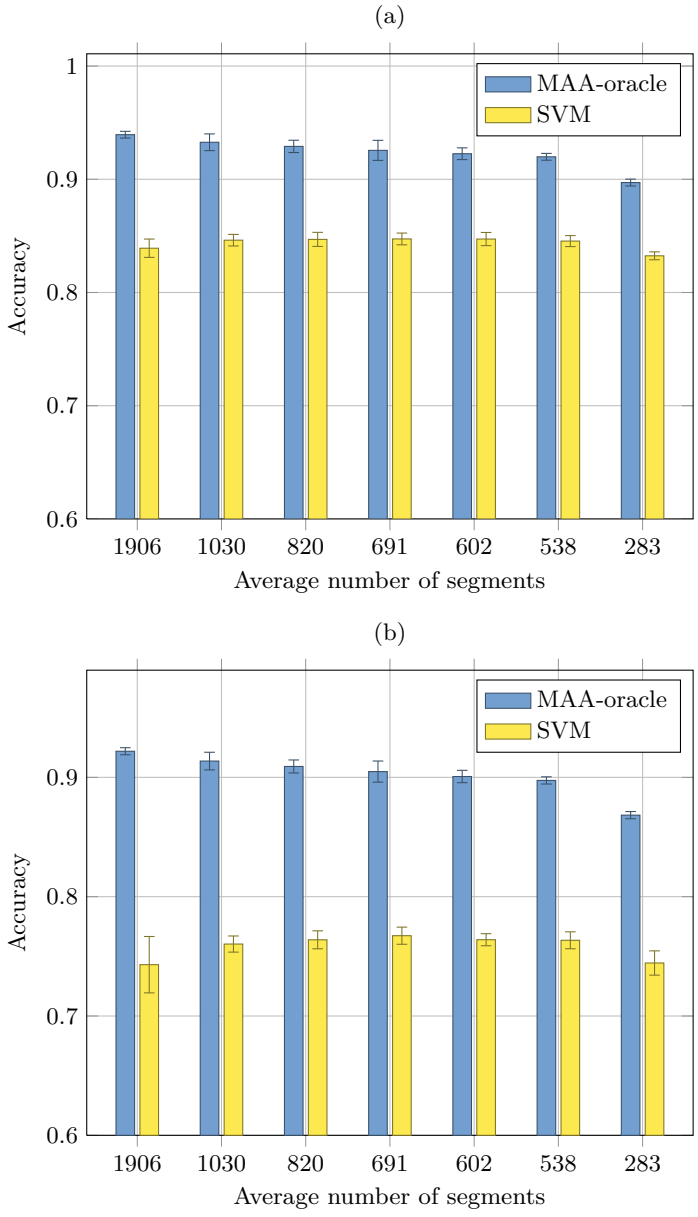


Figure 5.4: The effect of segmentation coarseness on (a) pixel-wise and (b) class-wise accuracy of the oracle and SVM classifier on the ECP dataset.

	Trained on		Evaluated on
	positives	negatives	
windows	3924 from Belgian facade images	8343 from PASCAL VOC	eTRIMS/ECP
doors	447 from Belgian facade images	8343 from PASCAL VOC	eTRIMS
cars (frontal)	516 front- and rear-view car images	4268 from PASCAL VOC	eTRIMS
cars (side)	344 from [107]	4268 from PASCAL VOC	eTRIMS

Table 5.1: Overview of the data used to train the generic detectors.

from a second source, but also allow our final labeling not to be constrained by the initial segmentation boundaries. This is especially apparent for the case of window detection, where the initial object boundaries often do not coincide with image gradients.

From all classes present in the ECP and eTRIMS datasets, some are best discriminated by their texture and color (*sky*, *grass*, *road*, ...). Other classes, such as *window*, *door* and *car*, are characterized by their distinctive shapes and sizes, and can therefore be discovered by classical object detectors. For these 3 object categories we trained object detectors, explained in more detail later, with training data coming from different sources. The total number of windows in the ECP dataset is large enough to train a detector which is **specific** to the Haussmannian style. Training a style-specific detector also benefits from the fact that Haussmannian windows and doors samples do not show much variance in appearance. In contrast, the eTRIMS dataset does not follow a fixed architectural style and shows a high variance of object appearances. At the same time, eTRIMS contains fewer samples per object class (1016 for windows, 85 for doors and 67 for cars). The higher variation combined with fewer examples makes training reasonable detector models by using only eTRIMS data infeasible. Therefore, we used data from an outside source to train style-agnostic window, door and car detectors. We call these detector models **generic** models (as opposed to **specific** models), as the data used for training (**generic** data) does not follow any specific style. As shown in Table 5.1, the window, door and car samples originate from various sources, e.g. from a dataset of Belgian facades, or from a general-purpose car datasets [107].

5.5.1 Object Detectors

Selecting the appropriate detector is not a simple task, as object detection is still an area of very active research. In recent years, many top-performing object detection systems have been based on the well-known deformable parts models (DPM) from Felzenszwalb *et al.* [46]. These detectors show excellent detection quality as demonstrated, for example, on the yearly PASCAL VOC [45] challenge. Using multiple components and parts gives these detectors an advantage when

detecting object classes characterized by a considerable amount of variation in their spatial extent. Conversely, when the object class is characterized to contain roughly rigid elements, classifiers based on a single template seem to be more appropriate. Lately, approaches based on the integral channels classifier [40] have demonstrated excellent quality [11] and detection speed [10]. The latter detector, dubbed `Very Fast` by the authors, not only reaches 100 Hz on the task of pedestrian detection, but also generalizes well to other classes. For example, in the German traffic sign detection challenge [80], one of the winning approaches [118] was based on this detector.

We decided to compare the `Very Fast` and the DPM detector for the window detection task, using the following setup. For both detectors we train one model in each of the 5 folds of the Haussmann-specific window training data, as described in Sec. 5.3. For each fold we use all available positive training samples, while patches not overlapping with windows are used as negative examples. Additionally, we augment the negative set with 8383 images not containing windows from the PASCAL VOC dataset. Speed comparisons were performed on an Intel Core i7 870 CPU + Nvidia GeForce GTX 590.

Deformable part-based model detector (DPM): The DPMs are trained using the latest release (version 5) [63] with default settings. The number of components is set to 1. The training took roughly 5 hours, and the testing speed of 3.8 sec/image can be sped up by a factor of $10 - 15$ by using a cascade [47]. We noticed that training this window detector with 2 or more components only reduces the overall quality while increasing the training time.

`Very Fast` detector: We use the publicly available open source implementation of the `Very Fast` detector [10]. The training is initialized by using a feature pool size of 30000 random features. We perform 4 rounds of training (2000 stage classifiers), where each round is followed by bootstrapping 5000 hard negative samples. With this setup, training lasts around 8 hours. The testing time of 2.1 sec/image can be sped up by a factor of around 40 by approximating nearby scales and using a soft cascade, as described in the original paper.

The performance of all detectors is evaluated using the PASCAL VOC overlap criterion of 50% overlap over union. Fig. 5.5 compares the mean detection rates of windows in the ECP dataset, where the detectors have been trained with Haussmannian windows. For each of the 5 folds, we trained a DPM detector and a `Very Fast` detector. All detectors are evaluated on their appropriate testing sets and the results are then averaged over the 5 cross-validation folds.

The results reveal that the single template-based `Very Fast` detector performs better than DPMs on this task. This behavior may be explained with the fact that the window and door classes do not consist of independently moving parts.

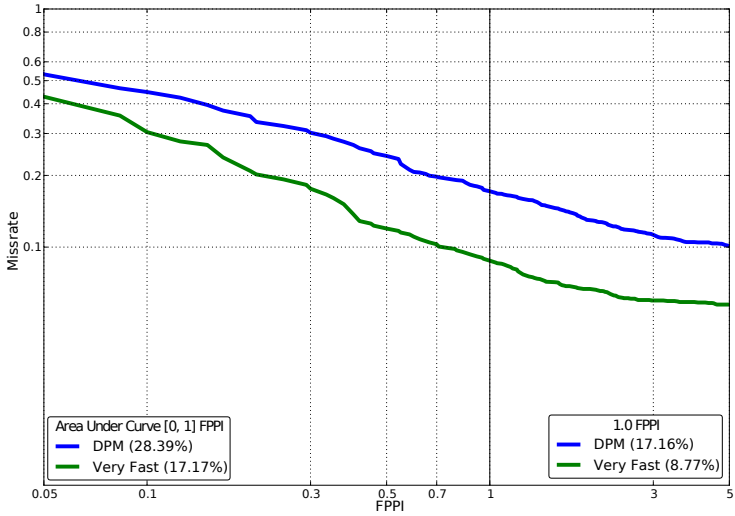


Figure 5.5: Comparison of the dataset-specific DPM and `Very Fast` on the task of window detection. The plot shows the mean FPPI versus miss-rate, averaged over 5 folds.

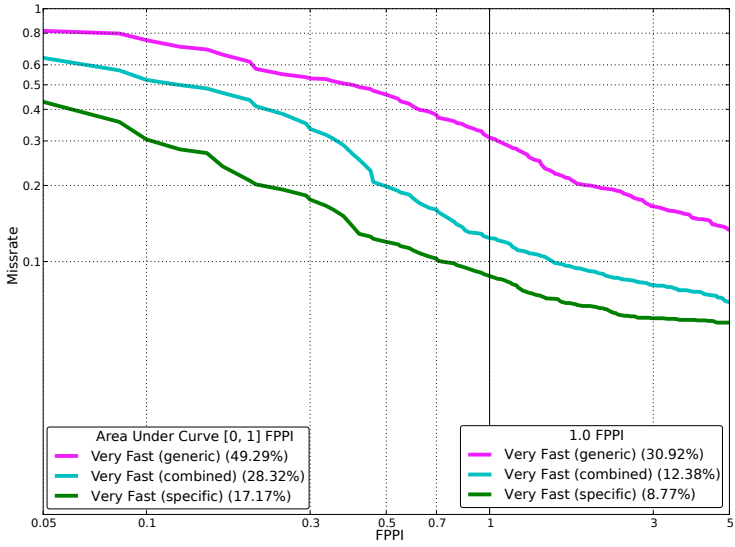


Figure 5.6: Comparison of specific, generic and combined window detector on the ECP dataset. The combined detector is trained on the joint training set of style specific and generic window samples.

Furthermore, image rectification leads to axis-aligned window corners. Due to the better detection quality and speed we opted for the `Very Fast` detector in all following detection experiments. Fig. 5.7 shows some example window detections of the `Very Fast` detector. For the task of car detection, DPMs might have a better performance due to the higher shape variability of the car class, but our experiments indicate that the `Very Fast` detector performs adequately on the few car samples in the eTRIMS dataset, where cars are usually shown either from the side, front or rear.

5.5.2 Generic and Specific Object Detectors

The ECP dataset contains 3096 windows and 109 doors, exhibiting the style of typical Haussmannian facades. Hence, all windows and doors have similar appearances and are therefore well suited to train Haussmann-specific window and door detectors. On the other hand, eTRIMS provides only 1016 window and 85 door instances from many different architectural styles, which leads to a high variance of e.g. appearance and aspect ratio. A common strategy to handle such diverse object classes consists in clustering the data into subsets (e.g. by their aspect ratio [46]) and independently training one detector for each subset. This would further reduce the number of samples used for training. We therefore did not train dataset-specific detectors on eTRIMS. Instead, we use the eTRIMS dataset as a proof of concept which shows that, even when using generic detectors trained with data coming from different sources, we can improve the labeling quality in our middle layer.

To recapitulate, we use style-specific detectors if there are enough samples in the training data. Otherwise, we train generic detectors. Still, we can gain some insight by evaluating the performance difference between detectors trained on style-specific and generic input data.

In Fig. 5.6 we compare the generic and specific window detectors on the ECP dataset. At 1 false positive per image (FPPI), the generic detector discovers around 70% of the windows, while the specific detector finds more than 90%.

Even though the generic detector could be used to improve window labeling in the middle layer, the specific detector has much better detection rates with fewer false positives. On the other hand, the advantage of using a generic detector lies in reduced training times when the system should be applied to many different styles. Instead of always retraining style-specific detectors - and having to know which style is relevant for any individual building - one might opt for collecting a large set of generic detectors for different facade elements and just select which detectors to use for specific styles. The detector obtained by **combining** specific and generic training data outperforms the

Figure 5.7: Example detections of the `Very Fast` specific window detector. The color encodes the confidence of the detection from high confidence (red) to low confidence (black).

generic detector, however it does not match the quality of the specific detector. By adding specific Haussmannian windows to the training data, we get a better representation of Haussmannian windows and therefore improve over the generic detector model. On the other hand, by adding generic window samples to the Haussmannian samples, we add a much higher variability to an otherwise quite homogeneous training data. We believe that this variability cannot be exploited, as the more general window detector introduces new false positives rather than detecting additional windows that were missed before. In conclusion, style-specific detectors perform better than generic detectors, especially when the data variation is limited (e.g. for Haussmannian windows). Generic detectors can still be used when it is infeasible to re-train detectors for every new style or when insufficient style-specific training data is available.

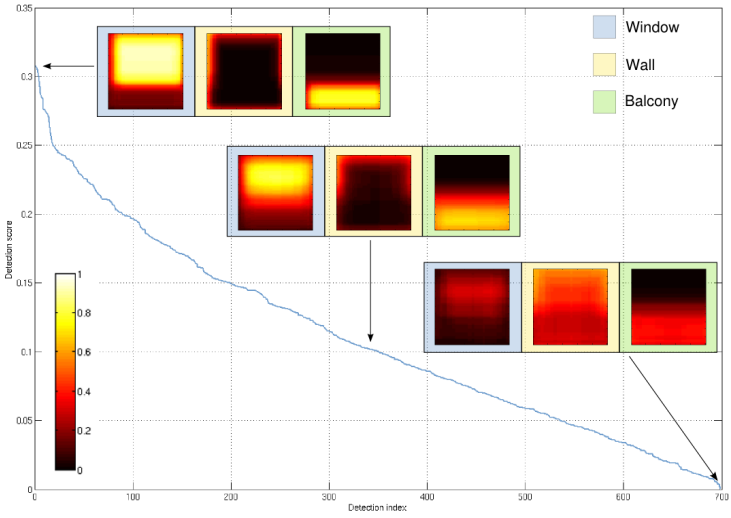
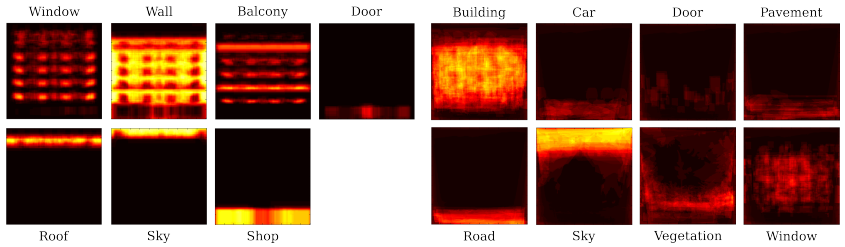


Figure 5.8: Learning label distributions for the window detector. Window detections on the ECP validation set are sorted by their score in descending order. High-scoring detections (top-left) provide a much stronger prior than the low-scoring detections (bottom-right)

5.5.3 Learning Detector Label Distributions

In order to merge the information coming from the object detectors with a semantic labeling of an image, we need to transform the detector output (typically a set of bounding boxes with scores) into per-pixel label probabilities. The simplest approach would be to simply set the probability of each pixel within a detection to 1 for the class corresponding to the detector (e.g. window), and zero to all other classes. However, window detections often cover other parts of the facade, such as a balcony or wall. The classification accuracy of balconies and walls would thus be negatively affected. To illustrate this, Fig. 5.7 shows an example output of the window detector, where the score of each detector is color-coded (brighter means higher detector confidence). Furthermore, not all detections ought to have the same influence: we want to significantly boost window probabilities in bounding boxes of high-scoring detections, but to be more conservative with low-scoring detections, since they might be false positives.

Let $\Delta = \{\delta_k\}_{1 \leq k \leq K}$ be the set of K detectors. In the ECP dataset we use only a window and a door detector, so $K = 2$. We propose a novel way of learning the detector label distributions $\mathbf{P}^{\delta_k}(l|x_i)$, i.e. the probabilities that a given point x_i in a test image belongs to one of the semantic labels $l \in \Psi$ according



(a) ECP (b) eTRIMS

Figure 5.9: Learned label maps from the training set in one cross-validation fold, by averaging over the different facades. Brighter colors denote higher label probability. Note the high level of regularity and alignment in the ECP dataset compared to the more washed-out probabilities for eTRIMS.

to the detector δ_k. To achieve this, we investigate how detections of a certain score spatially overlap with the ground truth labeled images in the validation set.

Let us denote the set of N images in the validation set with $X^v = \{\mathbf{x}_n\}_{1 \leq n \leq N}$, and their corresponding ground truth labeled images with $Y^v = \{\mathbf{y}_n\}_{1 \leq n \leq N}$.

After running a detector δ_k on the validation set, we obtain a set of M_k detections

$$D_k^v = \{d_j \mid \ d_j = (\mathbf{b}_j, r_j, \mathbf{y}_j)\}_{1 \leq j \leq M_k} \tag{5.1}$$

where each detection d_j is characterized by its bounding box $\mathbf{b}_j$, score (detector confidence) r_j and the labeled ground truth corresponding to the image where the detection was found: $\mathbf{y}_j \in Y$. The detections in the set D_k^v are sorted by their score in descending order.

Then, for each detection $d_j \in D_k^v$, we create a sub-image $\mathbf{S}_j$ by extracting the area of the corresponding ground truth label $\mathbf{y}_j$ covered by the bounding box $\mathbf{b}_j$, denoted as

$$\mathbf{S}_j = \mathbf{y}_j[\mathbf{b}_j] \tag{5.2}$$

The extracted sub-images are all subsequently rescaled to the same size using nearest-neighbor interpolation. For the normalized width u^{norm} and height v^{norm} we chose the value of 100 pixels, since most detection sizes in our dataset were on the same order of magnitude. The normalized sub-images are denoted as

$$\mathbf{S}_j^{norm} = NNResize(\mathbf{S}_j, v^{norm}, u^{norm}) \tag{5.3}$$

By construction, the normalized sub-images contain a subset of labels (classes) Ψ. Next, we create $|\Psi|$ binary label masks for each sub-image, defined as

$$\mathbf{B}_j^l = \mathbf{1}_l(\mathbf{S}_j^{norm}), \quad \forall l \in \Psi \tag{5.4}$$

where $\mathbf{1}_l$ is the indicator function selecting only pixels with the label l. To obtain a smooth label distribution, we average the binary label masks of detections with a similar score. For each detection, we consider γ neighboring detections in the original sorted list of detections. Let $j_0 = max(1, j - \frac{\gamma}{2})$. We define

$$\mathbf{Q}_j^l = \frac{1}{\gamma} \sum_{i=j_0}^{j_0+\gamma} \mathbf{B}_i^l, \quad \forall l \in \Psi \tag{5.5}$$

as the per-pixel probability that a given pixel in the bounding box $\mathbf{b}_j$ is labeled with l. The obtained $\mathbf{Q}_j$ is a valid probability distribution, since

$$\sum_{l \in \Psi} \mathbf{Q}_j^l = \mathbf{J}_{v,u} \tag{5.6}$$

where $\mathbf{J}_{v,u}$ is a matrix of ones. In our experiments we set $\gamma = 200$, as there are on average 700 detections in the validation set. Very small values of γ result in distributions that are no longer smooth, while by using higher values of γ the detection score starts to lose its effect, as all $\mathbf{Q}_j$ become rather similar.

Examples of the resulting $\mathbf{Q}_j$ are visualized in Fig. 5.8 for $l \in \{$*window*, *wall*, *balcony*$\}$ (other labels are not shown for clarity). For high-scoring detections (top-left corner of the image), our approach learns that the upper part of a detection should be assigned to the *window* label, while the lower part often corresponds to the *balcony* label. On the other hand, for lower-scoring detections, the effect of false positives firing on *wall* areas is so prominent that the *wall* label probability actually surpasses the *window* label. As we will see, the effect of false positives on the final labeling will be kept at bay, since our system hesitates to assign high label probabilities to low-scoring detections.

We can consider these learned label distributions as a look-up table during the testing phase. In a test image $\mathbf{x}^{test}$, we want to define the label distribution for each pixel, given the detections of a single detector δ_k. Initially, we assume no prior knowledge and assign a uniform label distribution to every pixel. Let

$$\mathbf{P}^{\delta_k}(l|x_i) = \frac{1}{|\Psi|}, \quad \forall x_i \in \mathbf{x}^{test}, l \in \Psi \tag{5.7}$$

be the initial probability distribution of labels in the image, where l is the predicted label for pixel x_i. After running the detector δ_k on the evaluation set, we obtain a set of M_k^e detections

$$D_k^e = \{d_j \mid \ d_j = (\mathbf{b}_j, r_j, \mathbf{y}_j)\}_{1 \le j \le M_k^e} \tag{5.8}$$

For every detection d_j in set D_k^e, we find the detection $d_{NN(j)}$ from the set D_k^v with the closest score to r_j. Its corresponding learned label distribution $\mathbf{Q}_{NN(j)}$ is resized to fit the bounding box $\mathbf{b}_j$, denoted as

$$\mathbf{Q}_{NN(j)}^{resized} = NNResize(\mathbf{Q}_{NN(j)}, v_j, u_j) \tag{5.9}$$

where u_j and v_j denote the width and height of the detection bounding box $\mathbf{b}_j$, respectively. Finally, the pixel probability distributions inside the bounding box are overwritten with the learned distribution, written as

$$\mathbf{P}^{\delta_k}(l|x_i) = Q_{NN(j)}^{resized}(x_i), \quad \forall x_i \in \mathbf{b}_j, l \in \Psi \tag{5.10}$$

The process is repeated for each detection d_j in the set D_k^e. Note that each position within $\mathbf{P}^{\delta_k}$ can be overwritten maximally once. There are no overlapping detections within one detector output due to non-maxima suppression. Detections of different object classes are handled by repeating the process for each detector δ_k, resulting in several learned priors $\mathbf{P}^{\delta_k}$.

5.5.4 Learning Facade Label Maps

In the previous section, we learned the label distributions only for pixels covered by detection bounding boxes. For other pixels, we assumed a uniform label distribution. However, it is logical to assume that the probability of a certain label also depends on the relative position of the pixel in the image. For example, one would expect *sky* pixels to appear mostly in the upper parts of the image, while the *shop* or *road* classes normally appear near the bottom of the image.

We can learn such a spatial prior in the form of *facade label maps* by analyzing the ground truth labels in the training set. First, we resize each ground truth image $\mathbf{y}_n$ from the training set Y^t to a common size ($u^{normf} = v^{normf} = 500$). The normalized ground truth image is defined as

$$\mathbf{y}_n^{norm} = NNResize(\mathbf{y}_n, v^{normf}, u^{normf}) \tag{5.11}$$

Similar to the previous section, we create $|\Psi|$ binary label masks defined as

$$\mathbf{C}_n^l = \mathbf{1}_l(\mathbf{y}_n^{norm}), \quad \forall l \in \Psi \tag{5.12}$$

which are then averaged over the training set, obtaining $|\Psi|$ facade label maps

$$\mathbf{R}^l = \frac{1}{N^t} \sum_{n=1}^{N^t} \mathbf{C}_n^l, \quad \forall l \in \Psi \tag{5.13}$$

where N^t is the number of images in the training set Y^t. Fig. 5.9 shows the learned label maps $\mathbf{R}^l$ for the ECP and eTRIMS datasets. The final distribution of labels in an evaluation image $\mathbf{x}^e$ with dimensions v^e and u^e is given by

$$\mathbf{P}^{\lambda}(l|x_i) = NNResize(\mathbf{R}_l, v^e, u^e), \quad \forall x_i \in \mathbf{x}^e, l \in \Psi \tag{5.14}$$

5.5.5 Incorporating Detector Knowledge into CRFs

In order to merge the labels coming from the bottom layer with those introduced by the detectors from the middle layer, we place a 2D Conditional Random Field (CRF) over the image pixels. Since we operate on pixels and not segments, the proposed model is quite different from the LOG-CRF model in Sec. 5.4.3, which is restricted to follow segment boundaries. We seek to minimize the CRF energy, defined as the weighted sum of unary potentials for each node and all pairwise potentials between neighboring nodes:

$$E(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \sum_{x_i} \Phi_s \left(y_i \mid x_i, \mathbf{w}\right) \tag{5.15}$$

$$+ \sum_{x_i} \sum_{x_j \sim x_i} \Phi_p \left(y_i, y_j \mid x_i, x_j, \mathbf{w}\right) \tag{5.16}$$

where x_i is an image pixel, $y_i \in \Psi$ represents the variable encoding the predicted label, $\mathbf{w} = \{w^{seg}, \mathbf{w}^{det}, \mathbf{w}^{lab}, w^{pair}\}$ is the set of CRF parameters, and the relation $\sim$ represents the 4-pixel neighborhood. We use the standard Potts model as the pairwise term, which encourages neighboring pixels to take on the same label. This has the effect of smoothing the output, with the degree of smoothing dependent on a parameter w^{pair}. The pairwise term is defined as

$$\Phi^p \left(y_i, y_j \mid x_i, x_j, \mathbf{w}\right) = \begin{cases} 0, & \text{if } y_i = y_j \\ w^{pair}, & \text{otherwise.} \end{cases} \tag{5.17}$$

The unary term is a linear combination of the low-level information from the segment classification, the learned prior facade label distributions, and the detector outputs:

$$\begin{aligned} \Phi^s(y_i \mid x_i, \mathbf{w}) \quad = \quad & -w^{seg} \log P^{\sigma}(y_i \mid x_i) \\ & - \sum_{k=1}^{K} \mathbf{w}_k^{det} \log P^{\delta_k}(y_i \mid x_i) \\ & - \mathbf{w}_{y_i}^{lab} \log P^{\lambda}(y_i \mid x_i) \end{aligned} \tag{5.18}$$

Here, P^{σ} is the per-pixel probabilistic output of the bottom layer. Since the classification in the bottom layer operates at the level of segments, all pixels within the same segment share the same probability. The detector potentials P^{δ_k} and prior facade label map potentials P^{λ} were defined in Sections 5.5.3 and 5.5.4, respectively. Parameters w^{seg},$\mathbf{w}^{det}$ and $\mathbf{w}^{lab}$ weigh the relative importance of segment classification, detector label maps, and facade label map priors. Note that $|\mathbf{w}^{lab}| = |\Psi|$, as we weigh each facade label map separately.

Applying the CRF model requires us to find the optimal labeling of a test image, given the set of parameters $\mathbf{w}$. The approximate solution to this problem can be found efficiently using graphcut-based methods [21], since our CRF model contains only unary and submodular pairwise terms. Additionally, due to the usage of the Potts model, the α-expansion minimization guarantees a solution that is within a factor of two of the global minimum [21]. In the next section we describe how the parameter vector $\mathbf{w}$ can be learned from the validation set.

5.5.6 Learning the CRF Parameters

There already exists a body of work on learning parameters in random field models. Most of these approaches use either a form of cross-validation or piecewise training. A good overview of parameter learning in CRFs can be found, for example, in [101] and [131]. We decided to follow the approach of Szummer *et al.* [173], which is an efficient technique of max-margin learning in grid graphs, e.g. images, based on the structured support vector machine [190]. This method represents parameter estimation as a maximum margin learning problem, formulated as

$$\begin{aligned} \underset{\mathbf{w}:\|\mathbf{w}\|=1}{max} \quad & \gamma \quad s.t. \\ E(\mathbf{y}, \mathbf{x}_n; \mathbf{w}) - E(\mathbf{y}_n, \mathbf{x}_n; \mathbf{w}) \quad & \geq \quad \gamma \quad \forall \mathbf{y} \neq \mathbf{y}_n \quad \forall n \end{aligned} \tag{5.19}$$

where $\mathbf{x}_n$ is an image, $\mathbf{y}_n$ its corresponding ground truth, and n indexes all instances in the training set.

The learning algorithm constrains the energy of the ground truth labeling $\mathbf{y}_n$ to always be smaller than any other possible labeling $\mathbf{y}$ by a margin γ. Since there is an exponential number of possible image labelings, it is not feasible to solve the problem formulated in Eq. 5.19. The solution proposed in [173] works with a much smaller subset of labelings $\{\mathbf{S}_n\}$, i.e. a constrained set. For each image, a lowest-energy labeling is found using an efficient method, such as graph cuts. If this energy does not satisfy the margin, the labeling is added to the subset $\mathbf{S}^{(n)}$. After all images are processed, the parameters $\mathbf{w}$ are updated to satisfy

the newly added constraints, and the process is repeated. Since there is only a finite number of labelings that can be added, the procedure is guaranteed to converge.

The above formulation is further improved by enforcing a larger margin when the labeling is far from the truth. This difference between desired and candidate labeling can be expressed in terms of a *loss function* $\Delta(\mathbf{y}_n, \mathbf{y})$. By adding slack variables ξ_n to account for constraint violations and rescaling the margin as proposed in [177], the following quadratic optimization problem is obtained:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{N}\sum_{n=1}^{N}\xi_n \quad s.t. \quad \forall \mathbf{y} \in \mathbf{S}_n \quad \forall n \tag{5.20}$$

$$E(\mathbf{y}, \mathbf{x}_n; \mathbf{w}) - E(\mathbf{y}_n, \mathbf{x}_n; \mathbf{w}) \geq \Delta(\mathbf{y}_n, \mathbf{y}) - \xi_n \tag{5.21}$$

$$\xi_n \geq 0$$

where C is the regularization parameter and N is the number of training images. A common approach is to use Hamming loss, i.e. the number of mislabeled pixels in an image, as the loss function. However, our datasets do not have a balanced distribution of classes, since some classes only constitute a small percentage of total pixels (e.g. the *door* class). Mislabeling the small classes does not significantly change the overall pixel accuracy, however the class-wise accuracy is severely reduced. Therefore, we modify the loss function to take into account the frequencies of classes in each ground truth image, producing a greater loss when a low-frequency label is misclassified, resulting in a weighted Hamming loss:

$$\Delta(\mathbf{y}_n, \mathbf{y}) = \sum_{i=1}^{|\mathbf{y}|} f^{-1}(y_i^n)[y_i^n \neq y_i] \tag{5.22}$$

where $[.]$ is the indicator function, and $f^{-1}(y_i^n)$ represents the inverse frequency of the label y_i^n in the ground-truth image $\mathbf{y}_n$.

To calculate the most violated constraint in Eq. 5.21 we must find the labeling y which minimizes the refined energy function E', defined as

$$E'(\mathbf{y}, \mathbf{x}_n; \mathbf{w}) = E(\mathbf{y}, \mathbf{x}_n; \mathbf{w}) - \Delta(\mathbf{y}_n, \mathbf{y}) \tag{5.23}$$

As shown in [173], the loss function can be 'absorbed' into the energy function if it decomposes the same way as the energy. Since the weighted Hamming

loss decomposes over image pixels (nodes in the CRF), we can transform it into an additional unary potential. This corresponds to augmenting the unary potentials in the CRF:

$$\Phi_s^{'}(y_i \mid x_i, \mathbf{w}) \quad = \quad \Phi_s(y_i \mid x_i, \mathbf{w}) - f^{-1}(y_i^n)[y_i^n \neq y_i] \tag{5.24}$$

where Φ_s is defined in Eq. 5.18. The resulting problem of minimizing E' can still be solved efficiently using α-expansion, as the energy remains submodular. Every labeling that violates the margin constraint in Eq. 5.21 is added to the constraint set $\mathbf{S}_n$, and the parameter vector $\mathbf{w}$ is updated by minimizing Eq. 5.20. The process is repeated until $\mathbf{w}$ remains unchanged. We have implemented this approach using the SVM^{struct} software [190].

5.6 Top Layer: Using Weak Architectural Principles

The previous two layers propose a generic approach to semantic labeling, which is initially based on super-pixel classification and subsequently enriched by object detectors. Although the results of the first two layers are quantitatively convincing, the effect of the initial segmentation is still present in the output. This manifests itself in the jagged boundaries of some elements as well as the missing or misplaced facade elements. Hence it is difficult to use the output of these layers to derive convincing facade models with clearly defined boundaries and structures. Therefore, in the top layer we add meta-knowledge about buildings without defining a full facade grammar, in contrast to Teboul *et al.* [181]. This meta-knowledge is expressed through the concept of *weak architectural principles*.

An important advantage of these guidelines over procedural grammar rules is that the former are directly observable in the images, whereas the latter keep some concepts implicit. Even if the combined application of a number of grammar rules may lead to, for example, vertical alignment of windows, there might be no single rule explicitly prescribing such alignment. An issue with style grammars can therefore be the indirect coupling between what they specify and what can be easily verified in the images. Our approach also enables modeling of irregular facades, as we use architectural concepts as guidelines, not as hard constraints. Some of the proposed principles are quite generic and can be re-used for many different facade styles, while others were intentionally designed with a certain style in mind, e.g. the Haussmannian style. Similar to object detectors, most principles are formulated for the objects in the facades

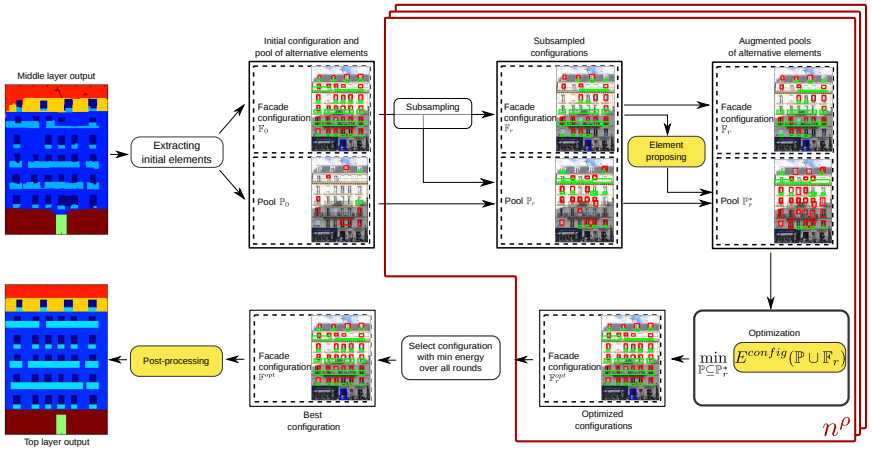


Figure 5.10: A high-level overview of the top layer. The blocks highlighted in yellow depend on weak architectural principles, see text for description.

(window, balcony, door), as these elements have a clearly defined boundary. In the end, the interplay between data evidence and various principles will influence the placement, modification or removal of facade elements.

5.6.1 Overview

Our first task is to define how the idea of weak architectural principles can be integrated into a generic system which allows for easy modification and addition of these principles. Therefore, we employ a modular design, where each principle has a well-defined interface and may be individually activated depending on the dataset at hand.

Fig. 5.10 shows the overview of our proposed system. The first step is to generate proposals of facade elements (bounding boxes with corresponding labels) from the output of the middle layer (Sec. 5.6.1). Let us define a facade configuration $\mathbb{F}$ as a set of facade elements which constitute a valid facade (i.e. no overlapping windows). The most probable interpretation of the facade from the previous layer is selected as the initial facade configuration $\mathbb{F}_0$, while non-selected elements (such as overlapping windows) are placed in a set of alternative facade elements, or a *pool* $\mathbb{P}_0$.

The initial configuration is not necessarily the correct one, as it might contain false positives. To remove them, we perform random subsampling, retaining a subset of elements in the configuration, and moving the rest to the pool of

alternative elements (Sec. 5.6.1). The subsampling is repeated in n^ρ rounds to increase the likelihood that, in at least one round r, the subsampled configuration $\mathbb{F}_r$ contains only true positives. Based on the subsampled configuration $\mathbb{F}_r$, the pool $\mathbb{P}_r$ is extended by new facade elements (Sec. 5.6.1). An optimization method is proposed to select the subset of elements in the augmented pool $\mathbb{P}_r^*$ which best complements the subsampled configuration $\mathbb{F}_r$ (Sec. 5.6.1), given an energy function E^{config}. The best facade configuration $\mathbb{F}^{opt}$ over all n^ρ is then fed into a post-processing step (Sec. 5.6.1).

The weak architectural principles are used for three different purposes in our system, see Table 5.2. First, they can *propose* new elements (Sec. 5.6.2). Second, some principles *grade* the proposal configurations through E^{config} (Sec. 5.6.2). Finally, certain principles are used to *modify* the facade element in the post-processing step (Sec. 5.6.2).

Extracting Initial Facade Elements

Starting from the pixel-wise classification output of the middle layer, the first step is to generate the initial configuration of facade elements $\mathbb{F}_0$. This configuration should contain facade elements such as windows, doors, or balconies. More precisely, each element is determined with the bounding box and its corresponding label. However, our middle layer produces a labeled image $\mathbf{y}^{L2}$, as opposed to discrete elements. To generate facade element proposals, we start by using connected components in the label map $\mathbf{y}^{L2}$ and define a minimal bounding rectangle R_z around the z-th connected component.

This minimal bounding rectangle is often too large compared to the actual facade element. Some initial super-pixels float over the object's real boundaries, which leads to over-sized minimal bounding rectangles. To mitigate this problem, we adjust the edges of each rectangle R_z by maximizing the coincidence with the edges of the connected component. Each edge of the current rectangle is adjusted by shifting it pixel by pixel towards the center of the rectangle. Let D_z denote the number of pixels inside of R_z belonging to the connected component. We limit the search range with the constraint that the number of connected component pixels inside the new rectangle must not fall below τ^{init} percent of D_z. The threshold τ^{init} was set to 0.6 in our experiments. At each position of the element edge, we calculate the overlap between the edge and boundary pixels of the connected component, divided by the edge length.

We find at most two possible edge proposals per rectangle side. The first one results from the highest edge overlap with the connected component boundaries. The second proposal is added only if the ratio between its overlap and the highest edge overlap is above τ^{edge}, set to 0.75 in our experiments. The rectangle with

the best combination of edge proposals is added to $\mathbb{F}_0$, and all other combinations are added to the pool of alternative elements $\mathbb{P}_0$.

Sub-Sampling

The initial facade configuration $\mathbb{F}_0$ obtained by the approach described in the previous section can potentially suffer from errors such as missing or misplaced elements, and false positives. As our proposed architectural principles are not designed to remove elements, dealing with false positives require separate consideration.

Our approach to dealing with incorrect facade elements is to repeatedly sub-sample the starting facade configuration with the goal of achieving at least one configuration containing only correct elements. Furthermore, we do not discard the elements that are not sampled, rather, we move them to the pool of alternative elements for later consideration.

The sub-sampling is repeated in n^ρ rounds. In each round we randomly split $\mathbb{F}_0$ into two disjoint subsets: the elements from the first subset are kept as the facade configuration of the r-th round $\mathbb{F}_r$, while the other elements are added to the pool $\mathbb{P}_0$, constructing the pool $\mathbb{P}_r$. The split is performed element-wise by adding an element to $\mathbb{P}_r$ with probability p^{rem}, or to $\mathbb{F}_r$ with probability $1-p^{rem}$. We set $p^{rem} = 0.4$, which allows us to keep on average more than half of the initial elements while at the same allowing to remove different combinations of potentially incorrect candidates. In our experiments, the setting of $n^\rho = 20$ produced satisfactory results. Further increase in the number of rounds typically does not result in finding a better configuration. When reducing the number of rounds, the performance degrades gracefully, converging to the initial labeling defined by $\mathbb{F}_0$.

Proposing New Elements

Assuming that the configuration $\mathbb{F}_r$ contains only true positives (which should hold true for at least one round r), we have a strong cue for discovering facade elements which are not present in either $\mathbb{F}_r$ or $\mathbb{P}_r$. For example, we might search for elements similar to those in the current facade configuration.

At this point, we can plug in any weak architectural principle which has the property of proposing new facade elements. Depending on the configuration $\mathbb{F}_r$, different additional facade elements might be proposed in each round (see examples in Sec. 5.6.2). The facade elements proposed by these principles are then simply added to $\mathbb{P}_r$, resulting in the augmented pool $\mathbb{F}_r^*$.

Principle	Propose	Grade	Post-process	ECP	eTRIMS
(Non-)alignment: vertical and horizontal	-	✓	✓	✓	✓
Similarity of different *windows* of the same *facade*	✓	-	-	✓	✓
Facade symmetry	✓	-	-	✓	✓
Co-occurrence of elements	-	✓	-	✓	-
Door hypothesis: first floor, touching ground	✓	✓	-	✓	-
Vertical region order: {$shop^*$, $facade^+$,$roof^*$, sky^*}	-	-	✓	✓	-

Table 5.2: Weak architectural principles used to complement the segmentation results of the first two layers. A tick in the "Propose" column denotes that the principle is used to propose new facade elements. Some principles can be used to evaluate the fitness of the facade configuration, denoted with a tick in the "Grade" column. The "Post-process" principles can be used in the last step of the inference procedure to modify the existing facade elements. Last two columns indicate which principles are used for each of the datasets.

Optimization

Starting from an incomplete facade configuration $\mathbb{F}_r$ and an augmented pool of elements $\mathbb{P}_r^*$ in the facade, our goal now is to find the optimal facade configuration $\mathbb{F}_r^{opt}$ with regard to a certain energy function. We assume that the elements in $\mathbb{F}_r$ are fixed, so the optimization amounts to the search for the optimal subset of elements in $\mathbb{P}_r^*$ which, combined with the entire set $\mathbb{F}_r$, minimizes the energy function:

$$\begin{aligned} \mathbb{F}_r^{opt} =& \mathbb{F}_r \cup \underset{\mathbb{P} \subseteq \mathbb{P}_r^*}{\operatorname{argmin}} \; E^{config}(\mathbb{P} \cup \mathbb{F}_r) \\ s.t. \quad & c_{\text{overlap}}(\mathbb{P}) \end{aligned} \tag{5.25}$$

The c_{overlap} constraints disallow any pair of overlapping elements in $\mathbb{P}$ to be selected at the same time, and can be expressed as a set of linear inequalities of the form $\mathbf{Ax} \leq \mathbf{1}$.

Selecting a subset of elements can be viewed as a binary integer optimization problem, where each variable indicates whether the corresponding element is included in the subset. In general, binary integer programming is NP-complete [87]. There are of course certain subsets of energy functions for which this optimization can be performed efficiently. For example, Kolmogorov and Zabih [92] show that if the energy function can be written as a sum of functions of up to two binary variables at a time (unary and regular pairwise potentials), the optimization can be performed in polynomial time. However, we allow the energy function to depend on an arbitrarily complex set of weak architectural principles, see Sec. 5.6.2. For this reason, and to keep the optimization as general as possible, we assume no prior structure of the energy function. This

rules out the use of deterministic optimization approaches, such as cutting plane methods (our objective function need not be convex), branch-and-bound (no knowledge on lower and upper bounds), or dynamic programming (no optimal substructure property).

Therefore, our choice is limited to (meta)heuristic methods. The simplest approaches, e.g. hill climbing or coordinate descent, are prone to getting stuck in local minima [147]. Monte Carlo methods such as simulated annealing [90] or MCMC [61] are more powerful, but typically require a large number of objective function evaluations. Genetic algorithms [78] are another popular metaheuristic, which was adapted for efficient solving of integer optimization problems by Deep *et al*. [37]. Although there is no proof of convergence, the latter implementation was shown to compare favorably to random search or annealing-based algorithms on certain datasets. We use an existing implementation of this approach in MATLAB's Global Optimization Toolbox [119], with default parameters, to solve the minimization problem in Eq. 5.25.

Post-Processing

After n^{ρ} rounds of sampling and optimization, the facade configuration with the lowest energy F^{opt} is selected as the best one. Note that the bounding boxes of facade elements boxes are fixed during optimization. Therefore, we employ post-processing principles on the best configuration, to clean up the final result by adjusting facade element boundaries.

5.6.2 Weak Architectural Principles

The weak architectural principles introduce meta-knowledge about facades into the labeling process. All principles take a configuration of existing facade elements as input, and can be divided into three main categories based on their output. The first category contains principles which propose new facade elements. These are used for generating new objects, which have not yet been discovered in the first two layers of our pipeline. Second, some principles can be used to grade proposal facade configurations, producing a single number, the 'energy' of the configuration as output. For example, the alignment principle should produce low energy for configurations with well-aligned elements. Third, some principles are used as a simple post-processing step, modifying existing elements in the facade configuration. Table 5.2 shows an overview of our proposed principles, sorted into the three main categories. The last two columns denote whether the principle was used while analysing a certain dataset. In

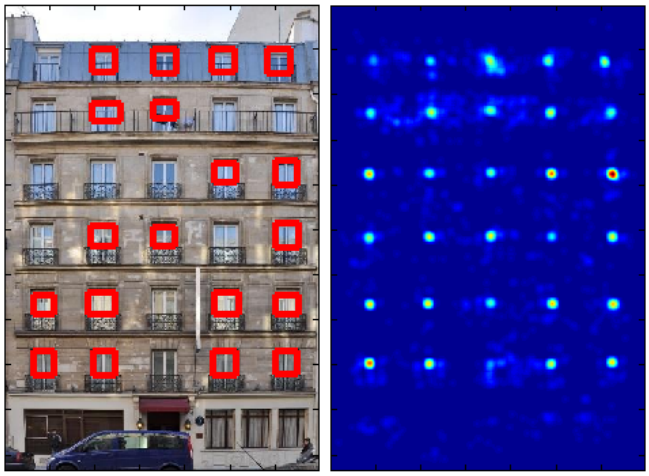


Figure 5.11: Similarity principle: Left: windows marked with red rectangles are the initially discovered windows. Right: the similarity voting space contains strong peaks at previously undetected windows.

the following sections, we describe in detail the aforementioned categories of principles.

Element-Proposing Principles

Based on a given facade configuration $\mathbb{F}_r$, element-proposing principles suggest new facade elements by exploiting meta-knowledge about (style-specific) facade structure.

As shown in Table 5.2, we identify three different principles for the proposal of new facade elements, namely the *similarity*, *symmetry* and *door hypothesis*. Each of these principles proposes a separate set of facade elements, which we denote $\mathbb{W}^{sim}$, $\mathbb{W}^{sym}$, and $\mathbb{W}^{door}$, respectively. Other principles can be added if necessary. We denote with $\mathbb{W}$ the set of all facade element proposals generated by the principles, i.e. $\mathbb{W} = \{\mathbb{W}^{sim} \cup \mathbb{W}^{sym} \cup \mathbb{W}^{door}\}$.

The similarity principle is based on the observation that most facades contain visually similar objects. If some elements are missing in the current facade configuration, they can still be found through visual similarity to the existing elements, see Figure 5.11. This principle is applied separately per object class and is parameterized by the median width u^{med} and height v^{med} of the object category. Our implementation is similar to the one described in Chapter 4.

Every object in the facade votes for similar elements using an ISM-like voting scheme [108]. As features we use self similarity descriptors [153] calculated at Harris corner points.

Let us consider a set of feature points that fall within the bounding box of a single element in the facade configuration. Each of these feature points is defined by its descriptor, a vote vector to the center of the bounding box, and the size of the bounding box of the element. For each feature point, we search for 10 nearest neighbors among all feature points in the image based on its descriptor. The neighbors then cast votes into a global voting space using a Gaussian kernel of size $\min(u^{med}, v^{med})$. The process is repeated for all facade elements in the configuration. After all votes are collected, we perform greedy non-maximum suppression: each maximum defines an area of size $u^{med} \times v^{med}$ in which we keep the maximum and set the other values of the voting space in that area to 0. Most of the maxima in the voting space will be situated inside the bounding boxes of existing elements. Each remaining maximum defines a bounding box with the size defined as the median of bounding boxes sizes corresponding to votes which contribute to the maximum.

As the similarity voting is performed based on the subset $\mathbb{F}_r$ (Sec. 5.6.1), some maxima will correspond to facade elements already in $\mathbb{F}_r$. Only new elements build the set $\mathbb{W}^{sim}$. We limit the number of new proposals to $|\mathbb{F}_r|$, as we do not wish to add more proposals than the number of elements currently in the configuration.

Harris corners are also used as a simple measure in the principle of vertical **symmetry**. The interest points are mirrored about a symmetry axis (line) hypothesis. A match is defined by two interest point locations, which are mirrors of each other about the symmetry axis. Note that we only match interest point locations at this point, not their descriptors.

The maximum number of matches divided by the points under consideration defines a simple symmetry score for the corresponding symmetry axis. If symmetry is detected (symmetry score $> \tau^{sym}$), facade elements are mirrored about the similarity line with the maximum score and constitute the set of proposals $\mathbb{W}^{sym}$. For the value of τ^{sym} we select the lowest symmetry score from all symmetric facade examples in the training and validation sets. Fig. 5.12 shows an example of a symmetric facade, with the symmetry axis denoted as a dashed blue line.

The **door hypothesis** principle creates a single door proposal $\mathbb{W}^{door}$, and it is only applied when $\mathbb{F}^r$ does not already contain a door bounding box. If there are no door objects in the pool of alternative elements either, we fall back to the probabilistic output of the bottom layer. We expect a door to be at least

the size of a median window in the facade. Therefore, we first search for the maximum response by sliding a window of size $u^{med} \times v^{med}$ (median window size) over the bottom layer probabilistic output and averaging pixel probabilities corresponding to the *door* class inside that bounding box. From the position with the maximum support, we greedily grow the door bounding box until the average probability of the *door* class starts decreasing. Even if the real image contains several doors, this principle is limited to produce only one element, the one with higher support.

Element-Grading Principles

Element-grading principles contribute to the energy function E^{config} which is used to judge a proposal facade configuration $\mathbb{F}$. We define this energy function as

$$E^{config}(\mathbb{F}; \mathbf{y}^{L2}) = E^{data}(\mathbb{F}; \mathbf{y}^{L2}) + \sum_{\pi \in weakPrinciples} \alpha^{\pi} E^{\pi}(\mathbb{F}; \mathbf{y}^{L2}) \tag{5.26}$$

where $\mathbf{y}^{L2}$ represents the middle layer output (CRF labeling). The data term E^{data} encourages the configuration to be as similar as possible to the prediction from the middle layer. It is independent from any principle and defined by:

$$E^{data}(\mathbb{F}; \mathbf{y}^{L2}) = \sum_{l \in \Psi_{obj}} E_l^{data}(\mathbb{F}; \mathbf{y}^{L2}) \tag{5.27}$$

$$E_l^{data}(\mathbb{F}; \mathbf{y}^{L2}) = -\sum_{y_i \in \mathbf{y}^{L2}} [y_i = l \wedge g(y_i, \mathbb{F}, l) = 1] / \sum_{y_i \in \mathbf{y}^{L2}} g(y_i, \mathbb{F}, l) \tag{5.28}$$

$$+ \sum_{y_i \in \mathbf{y}^{L2}} [y_i = l \wedge g(y_i, \mathbb{F}, l) \neq 1] / \sum_{y_i \in \mathbf{y}^{L2}} [y_i = l], \tag{5.29}$$

where $\Psi_{obj} \in \Psi$ denotes the subset of all object labels, as only facade objects are optimized in this step. Note that we define the data term separately for each object label $l \in \Psi_{obj}$. The function $g(y_i, \mathbb{F}, l)$ returns 1 when y_i is covered by a bounding box with the same label l from $\mathbb{F}$. Expression 5.28 reduces the energy when labeled pixel y_i is covered with a facade element with the same label from configuration $\mathbb{F}$, while Eq. 5.29 penalizes object pixels not covered by facade elements from $\mathbb{F}_r$.

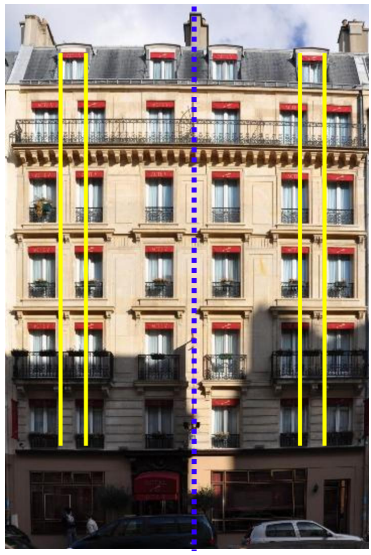


Figure 5.12: The (non-)alignment principle states that facade elements should be either aligned or clearly off-center. In this image, windows exhibit a high degree of horizontal and vertical alignment. Two windows bordered with yellow lines are vertically off-center to other windows. This should not be penalized, as this is a often-observed window configuration. The blue dashed line depicts the symmetry axis of the facade.

The energy of each grading principle is weighted by α^{π} and added to the total energy. We determine the values for α^{π} on the validation set. In the following, we will describe the principles that contribute to the energy function.

The (non-)alignment principle is based on the observation that many facade elements of the same type are either exactly aligned or clearly off-center (see the yellow lines in Fig. 5.12). The energy for an object class is defined as

$$E^{align}(\mathbb{F}) = \sum_{(e_1,e_2)} \Big(\beta(s_1^{(e_1)} - s_1^{(e_2)}; \tau^w) \quad + \quad \beta(s_2^{(e_1)} - s_2^{(e_2)}; \tau^w) + \tag{5.30}$$

$$\beta(t_1^{(e_1)} - t_1^{(e_2)}; \tau^h) \quad + \quad \beta(t_2^{(e_1)} - t_2^{(e_2)}; \tau^h) \Big) \tag{5.31}$$

$$\beta(z;\tau) = \begin{cases} \frac{\tau^2}{6}(1 - [1 - z/\tau]^2)^3, & \text{if } |z| \leq \tau \\ \frac{\tau^2}{6}, & \text{if } |z| > \tau \end{cases} \tag{5.32}$$

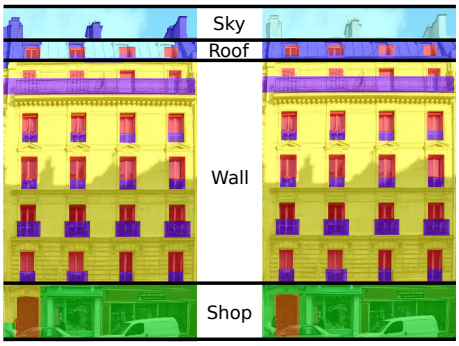


Figure 5.13: The vertical region order principle determines the border between the sky, roof, wall and shop areas of the facade.

where e_1 and e_2 refer to each possible combination of same-class elements in $\mathbb{F}$, and (s_1, t_1) and (s_2, t_2) represent the coordinates of the top-left and bottom-right corners of an element. The capped influence function β rates the top, bottom, left and right alignment of a pair of facade elements. The function has a constant value as soon as the distance between element boundaries exceeds a certain threshold τ. Based on our initial observation that windows are either aligned or completely misaligned, we set τ^w and τ^h to half of the median object width and height respectively.

The principle of **co-occurring elements** reflects the observation that pairs of elements appear in certain fixed configurations. One particular case of this principle is window and balcony co-occurrence: a facade should not have a balcony without a corresponding window. Therefore, we first try to assign at least one window to each balcony. Balconies without a corresponding window are then penalized by adding a constant value τ^{occ} to the energy term. By setting $\tau^{occ} > 0$ we increase the energy of solutions containing one or more solitary balconies. The co-occurence principle might as well be used for other pairs of elements or even as a facade element proposing principle, but we leave this for future work.

Post-Processing Principles

The **vertical region order principle** states the specific order of the *sky*, *roof*, *wall* and *shop* areas observed for Haussmannian facades. We enforce such an order in our output labeling (See Figure 5.13). First, we find the initial split lines between the aforementioned areas. This is done by finding the connected components of the corresponding labels and placing a split line on the lower

boundaries of the regions. Then, similar to Sec. 5.6.1, we test candidate split positions by moving the split lines pixel by pixel in the upward direction, seeking to maximize the overlap between the split line and region boundary pixels. After the splitting positions between the regions are found, we switch the labels of the aforementioned classes to be consistent with the region order.

The (non-)alignment principle is also used in the post-processing step. We use the second part of the energy function E^{align} (Eq. 5.31) to align windows horizontally by adjusting the upper and lower borders of their bounding boxes. We find the local minimum of this energy with the iterative BFGS Quasi-Newton method [50]. The result is that all windows aligned horizontally within a tolerance of τ^h will now be perfectly aligned with each other.

5.7 Results

We compare our approach to previous work on two datasets for facade parsing, see Sec. 5.3. Tables 5.3 and 5.5 show the performance of all approaches evaluated per class, as well as the average pixel and class accuracies. We show the results of our system for each layer of the pipeline together with the top layer performance of our previous work [113] and the performance of other approaches. Example output of our system can be found in Figures 5.15 and 5.16.

5.7.1 ECP Database

All methods were evaluated following the same 5-fold cross validation, evaluated on the updated annotations as described in [113]. In Table 5.3, we compare our results with the Random Forest (RF) pixel classifier used as a baseline in [182], the Reinforcement Learning (RL) grammar-based approach from Teboul *et al.* [181], the domain knowledge learning (DKL) work of Dai *et al.* [35], and the Spatial Pattern Templates (SPT) from Tyleček *et al.* [192]. We retrained and evaluated the RF and RL classifiers using the publicly available code, while DKL and SPT results were provided by their respective authors.

As expected, the simplest approach - Random Forest classifier based directly on image patches [182] - exhibits the poorest performance. This can be partly attributed to weak features (raw pixel values), and partly to the lack of context, since every pixel is classified based only on its local patch of size $13x13$. Compared to this approach, our bottom layer already achieves a better performance for all classes, due to the fact that we use a superpixel-based approach and more discriminant extracted features.

		Wi	Wa	Ba	Do	Ro	Sk	Sh	**Pixel average**	**Class average**
RF [182]		33	67	32	82	52	92	20	53.46	53.73
RL [181]		55	82	49	43	52	97	82	73.24	65.66
DKL [35]		72	87	70	66	80	93	91	83.50	79.80
SPT [192]		75	86	73	66	85	95	95	84.20	82.14
3Layer [113]		75	88	70	67	74	97	93	84.17	80.71
Ours	L1	64	91	75	41	82	94	91	84.75	76.67
	L2	76	90	81	58	87	94	97	**88.07**	83.36
	L3	78	89	87	71	79	96	95	88.02	**85.22**

Table 5.3: Performance on the ECP dataset (in percent). All experiments were performed with the same protocol (5-split cross-validation with 60 training, 20 validation and 20 testing images). L1, L2, L3: Our bottom, middle and top layer output, respectively.

The state-of-the-art grammar-based RL approach [181] requires a prior definition of a specific Haussmannian-style procedural grammar. The free parameters of the grammar are then optimized such that the agreement between the resulting labeling and the bottom-up merit function (RF labeling) is maximized. This approach greatly improves upon the results of the earlier RF approach, yet still performs worse than any layer output of our approach. One of the reasons for this behaviour is that the somewhat over-simplified grammar restricts the space of possible facade labelings, imposing certain structure even if it is not present in the image. For example, vertically misaligned roof windows are not supported with the existing grammar, and are thus mislabeled. Our approach does not suffer from these issues. One may argue that the lower performance of the RL method stems from their usage of less informative bottom-up cues. Therefore, we investigate how the RL approach performs when using much stronger merit functions, namely the output of our bottom and middle layers. The results in Table 5.4 (right) show that the RL method indeed benefits from stronger bottom-up information. However, when using our bottom and middle layer as the merit function, the RL method achieves lower performance than the merit function itself. This supports our claim that adding strong grammar constraints can actually decrease the overall performance.

Comparable results to our bottom layer were achieved by Dai *et al.* [35], an approach which, like ours, forgoes the usage of style-specific grammars. Instead, it is designed to adapt to various building styles by learning weights for different architectural principles. However, as the approach was tested on only one building style (ECP dataset), it is difficult to assess the effectiveness of the learning algorithm. Furthermore, their initial image segmentation into rectangular regions is fixed and might pose a problem when dealing with more

		Pixel average	Class average			Pixel average	Class average
Ours	L1	84.75	76.67	RL [181]	RF merit	73.24	65.66
	L2	**88.07**	83.36		L1 merit	82.41	72.58
	L3	88.02	**85.22**		L2 merit	83.10	76.17

Table 5.4: Comparison of our approach to the Reinforcement Learning (RL) approach of Teboul *et al.* [181] with different merit functions, on the ECP dataset. RF: Random Forest; L1, L2, L3: Our bottom, middle and top layer output, respectively.

general facades containing irregular appearance (e.g. eTrims dataset). Our top layer utilizes a more flexible set of principles, which are not restricted to follow the initial segmentation. For example, we allow classes such as *car* or *vegetation* to keep their irregular boundaries.

Another region-based method, SPT [192] achieves comparable results to our bottom layer, even outperforming it in class accuracy. This demonstrates that adding a region-based CRF with higher-order potentials on top of the initial segment classification boosts performance. We expect that our approach would benefit by integrating the SPT method in the first layer, which we leave for future work.

When considering the added value of each layer in our approach, it is clear that the middle layer produces the biggest improvement in pixel accuracy for the *window* and *door* classes, as was expected for the usage of object detectors. Additionally, the accuracy of other classes goes up due to the usage of learned label maps (Sec. 5.5.4) and the smoothing property of the CRF (Sec. 5.5.5). By introducing high-level knowledge through the top layer, we further improve on most of the classes. The noticeable drop in the *roof* and *sky* class can be explained by the fact that the "region order" principle (Sec. 5.2) imposes a straight line to separate regions which does not always match the real, more complex, boundary. We nevertheless kept these strict horizontal split lines between image regions, as they facilitate the process of procedural facade modeling.

Compared to the top-layer output of our previous work [113], we improve on almost all classes, boosting the average pixel accuracy to 88%. The increase of nearly 4% was achieved through several refinements, namely: using SVM as the region classifier, using stronger detectors, learning label priors, learning CRF parameters, and using the new top-layer sampling approach.

		Bu	Ca	Do	Pa	Ro	Sk	Ve	Wi	**Pixel average**	**Class average**
CRF [210]		71	35	16	22	35	78	66	75	65.80	49.75
HCRF [209]		67	36	14	85	53	80	78	80	69.00	61.63
ICFHGS- [53]		-	-	-	-	-	-	-	-	77.22	**72.23**
SPT [192]		89	70	37	64	68	81	84	68	82.10	70.13
3Layer [113]		86	67	18	35	47	91	81	80	80.81	63.20
Ours	L1	91	61	26	29	51	94	82	66	80.42	62.52
	L2	91	74	50	15	73	97	87	73	**83.39**	70.00
	L3	89	73	49	15	73	97	87	75	82.90	69.81

Table 5.5: Performance on the eTRIMS dataset. Class accuracies are shown in percent. The per-class results from Fröhlich *et al.* [53] were obtained on only one cross-validation fold, thus we do not report them. L1, L2, L3: Our bottom, middle and top layer output, respectively.

5.7.2 eTRIMS Database

As can be seen in Table 5.5, we outperform all previous results reported on the eTRIMS dataset in terms of overall pixel accuracy. It is important to note that even though some methods [53, 192] achieve higher class average, their pixel accuracy is still lower than ours. This can be explained with the poor performance of the *pavement* class in our approach, especially after the smoothing effect of our CRF, which increases the confusion with the bordering segments (mainly *road*). One of the reasons for this behaviour of the CRF is that its parameters are learned on a relatively small validation set (10 images), reducing the effect of unary potentials.

The difference between our bottom and middle layer is most apparent for the *window*, *car*, and *door* classes. These are the very classes for which we had trained object detectors. Additionally, the *road* class performance is significantly boosted due to the smoothing effect of the CRF (unfortunately, at the cost of the aforementioned *pavement* class). Finally, in the top layer, we only improve the performance of the *window* class, which is not surprising, as this is the only class in this dataset for which we use weak architectural principles. We also observe a 3% performance drop for the *door* and *building* classes. By analyzing the output data, we can see that the door accuracy drops due to the rectangularization process (see Sec. 5.6.1). Since some doors were partially covered by *window* detections in the middle layer, they were re-labeled as windows when defining rectangular window regions. Furthermore, many of the buildings in eTRIMS contain window shutters, which are annotated with the *building* class in the ground truth. Our generic detector, on the other hand, is trained on data which includes the shutters in the window structure, therefore increasing the confusion between the *window* and *building* class. Even though the final pixel

		Train	Test
Bottom layer	Segmentation	-	3.85 s
	Feature extraction	-	3.15 s
	Classifier	18 m	3.05 s
Middle layer	Detector	8 h	2.1 s
	Label maps	1 m	-
	CRF	70 m	3.5 s
Top layer	Subsampling and optimization	-	180 s

Table 5.6: Computation times for our method on the ECP dataset. 'Train': total time spent during training for each method. Items marked with '-' in the 'Train' row denote that the method has no training phase. 'Test': computing times for one test image. Note that label map learning is a learned prior, so it has no computing time during testing.

accuracy of the top layer is slightly lower than the accuracy of the middle layer, the resulting labeling is more visually pleasing, as can be observed in Fig. 5.16. Compared to our previous work [113], we notice a significant increase in the performance of almost all classes.

5.7.3 Computing Times

We performed all of our experiments on an Intel Core i7 870 CPU with 8 cores. Table 5.6 shows the average computing times on the ECP dataset, differentiated with respect to the different layers. Please note that the training phase differs for each method. As said in Sec. 5.4, the SVM classifier training is performed on the training set, while detector label maps (Sections 5.5.3 and 5.5.4) and the CRF parameters are learned on the validation set. The training protocol for detectors is described in Sec. 5.5.1.

5.7.4 Application: Image-Based Procedural Modeling

We use the output of the top layer in a straightforward procedural modeling scenario, encoding the facade as a set of CityEngine CGA rules [42]. The 7 different classes from the ECP dataset correspond to the terminal symbols of the procedural grammar. However, we make a distinction between facade element classes (*window*, *balcony*, *door)* and region classes (*wall*, *roof*, *sky*, *shop*). Each element class is modeled in a separate layer and then overlayed on the vertical split of facade regions. For example, we create the *window* layer by extracting the binary mask of windows from the output of our system. This binary mask is subdivided into rows and columns of elements, and encoded as a set of splitting

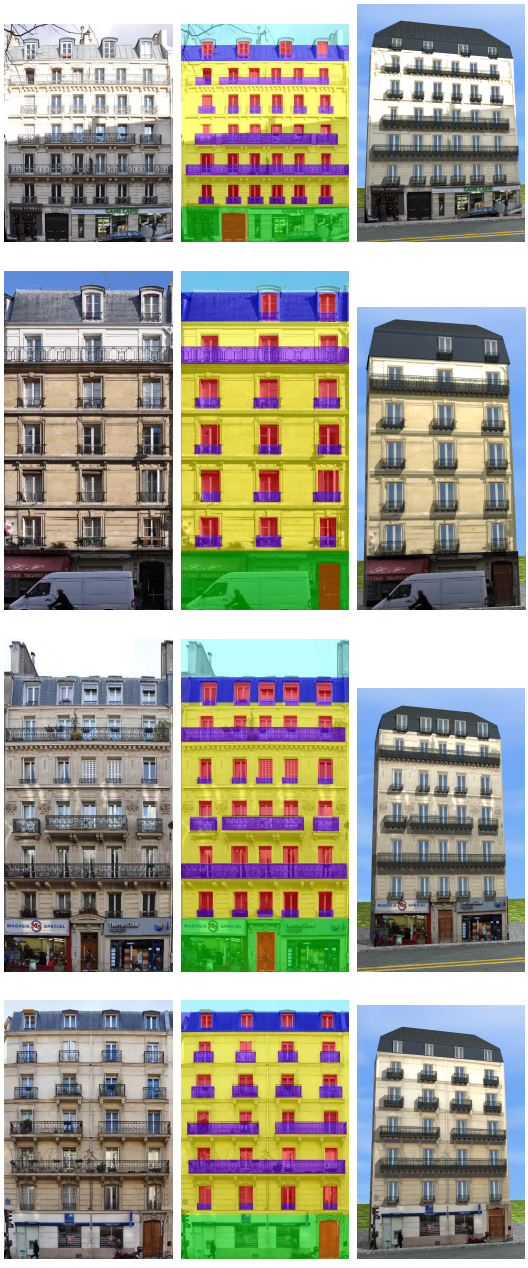


Figure 5.14: Results on the ECP dataset. (Left) Input image. (Middle) Our top-layer labeling. (Right) Procedural building model.

CGA rules. The terminal symbols are then replaced with 3D models from a library of architectural elements. Finally, the texture of the original image is projected onto the *wall* and *shop* area, to create a more realistic visualization. Several rendered models can be seen in Fig. 5.14. Note that our approach is able to handle non-aligned roof windows correctly (first row), while it is not forced to hallucinate non-existing ones (second row).

5.8 Conclusion

This chapter proposed a grammar-free method for facade parsing which is divided into three layers. For the bottom layer we explored a variety of different segmentation and classifier combinations to get our initial bottom up facade labeling. In the middle layer we then introduced the usage of object detectors to improve over the initial labeling. The results from the bottom and the object detector responses are combined in a principled way by using a CRF formulation, where the weights of the different CRF terms are estimated automatically. Finally in the top layer we added facade specific information via *weak architectural principles*. We proposed a general framework in which principles can be removed or added. This facilitates the usage of this layer for other facade styles. The output of our top layer are architecturally plausible facade structures with clearly defined boundaries and structures. Our method was evaluated on two datasets and shows state of the art performance.

In a final step, we demonstrate how the output of our top layer can directly be used for the image-based procedural modeling of facades. Instead of building our system upon a previously defined grammar – as demonstrated in the previous chapter – we could actually infer procedural rules from the output of our system. These rules are instance specific and if extracted from a single building can in that case only be used to generate a model of that building. In the following chapters, we will show that not only can we induce procedural grammars with generalizing capabilities from clean data, but also use the noisy output of the middle layer to infer hierarchical structures consistent across the entire dataset.

Figure 5.15: Results on the ECP dataset. (Left) The original image. (Middle-left/center/right) Outputs from the bottom, middle and top layers, respectively. (Right) Ground truth.

Figure 5.16: Results on the eTRIMS dataset. (Left) The original image. (Middle-left/center/right) Outputs from the bottom, middle and top layers, respectively. (Right) Ground truth.

Chapter 6

Semantic Segmentation of 3D Urban Scenes

In the previous chapter, we focused on an inherently two-dimensional task: parsing facades from images. In this chapter[1] we generalize our approach to the task of semantic segmentation of 3D city models. Starting from an SfM reconstruction of a street-side scene, we perform classification and facade splitting purely in 3D, obviating the need for much slower image-based semantic segmentation methods. We show that a properly trained pure-3D approach produces high quality labelings, with significant speed benefits allowing us to analyze entire streets in a matter of minutes. Additionally, if speed is not of the essence, the 3D labeling can be combined with the results of our 2D facade classifier, complementing the performance. Further, we propose a novel facade separation algorithm based on semantic nuances between facades. Finally, we upgrade the architectural principles used for 2D facade labeling to a set of new, 3D-specific principles and propose an efficient optimization scheme based on an integer quadratic programming formulation.

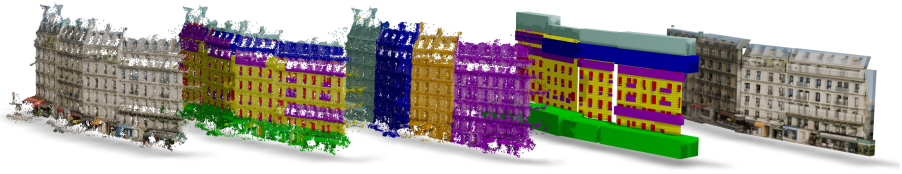


Figure 6.1: **The proposed approach for semantic segmentation of building blocks - overview.** In contrast to the majority of current facade labeling methods, our approach operates completely in 3D space. (a) image-based SfM 3D point cloud (b) initial point cloud classification (c) semantic facade splitting (d) structure modeling through architectural principles and (e) projected original images onto the estimated 3D model. The advantages of a pure-3D approach range from significant speed-up to complementarity with 2D classifiers.

6.1 Introduction

Increasingly, the topics of recognition and 3D reconstruction are intermingled. On the one hand, adding 3D features may aid recognition, on the other the knowledge about object classes helps with their 3D modeling. In the end, one can imagine feedback loops - cognitive loops if you will (e.g. [183])- where a system jointly evolves through the solution spaces that each such subproblem (e.g. recognition, 3D reconstruction) lives in. Human vision seems to strive for such kind of unified, consistent interpretation and the endeavour seems to serve us well.

This chapter focuses on leveraging the 3D information to aid the reconstruction of city models. The task comes with both the subtasks of recognition and 3D modeling. Thus, the models should not only consist of high-quality 3D models, but ought to come with delineated functional units (e.g. windows, doors, balconies, etc.). Although substantial effort has already gone into the creation of 3D city models, efforts to render those 'semantic' are rather recent. One of the most important steps to that effect is semantic facade parsing, introduced in the previous chapter. However, insted of treating the 2D labeling as a pre-processing step, this chapter investigates whether we can benefit from a direct coupling to the 3D data that mobile mapping campaigns also produce. More concretely, this chapter presents an entire pipeline for the analysis of building blocks, starting

[1]The chapter is based on the joint work with Jan Knopp, Hayko Riemenschneider and Luc Van Gool, to appear in CVPR 2015. While all parts of this work are a result of joint work and discussion, Anđelo Martinović and Jan Knopp worked on 3D classification and weak architectural principles in 2D and 3D, and share first authorship. Hayko Riemenschneider focused mostly on the facade splitting approach.

from raw images and resulting in a semantic 3D model, with all steps carried out in 3D. As we will show, the avoidance of jumping back and forth between 2D and 3D leads to substantially shorter runtimes (20x faster).

In particular, this chapter introduces three main contributions:

1. An end-to-end facade modelling fully in 3D;
2. A novel facade separation algorithm based on the results of semantic facade analysis;
3. A generalization of the previously introduced 2D weak architectural principles like alignment, symmetry, etc. to 3D, along with a faster optimization scheme.

6.2 Related Work

This chapter joins multiple research areas for the purpose of 3D facade understanding and modeling. Therefore, we briefly review the current state of the art in (1) 3D classification, (2) facade parsing and (3) facade separation.

6.2.1 3D Classification

A vast amount of work has dealt with the issue of 2D classification, see Chapter 5. However, the bulk of 3D classification work is rather recent, especially where 2D and 3D features are used together. To the best of our knowledge, Brostow *et al.* [23] were the first to combine image classification and sparse 3D points from SfM. Ladicky *et al.* [103] combine depth maps and appearance features for better classification. In a similar vein, some approaches [127, 5] combine LIDAR and image data.

Since then, recent works show the advantage of combining 3D and 2D for classification [165, 88, 85, 150, 212] or place recognition [149] using LIDAR, 3D CAD or Kinect [214] models. However, 3D is used in the context of improving 2D recognition, as these approaches still heavily rely on 2D features. Some even show that 2D information is much more important than 3D descriptors [150]. In contrast, instead of using 3D as useful aid for the 2D classifier, we design an exclusively 3D pipeline as a fast alternative to previous approaches, with competitive results.

In the 3D-only domain, a variety of local descriptors have been introduced in recent years. Unfortunately, the best performing features are typically expensive

to calculate, or limited to e.g. manifold meshes [86, 22, 91]. Furthermore, automatically obtained 3D is incomplete, containing noise, holes and clutter. Thus, spin images [84] are still a popular choice (shown to be robust in the presence of noise [148]), combined with several low-level features such as color, normals, histograms etc. [65, 120, 165, 130, 143, 172]. We follow this vein of research, carrying out facade labeling completely in 3D: from using simple 3D features, point-based classification with Random Forests, and with a 3D Conditional Random Field smoothing. This results in competitive results with significant speed benefits.

6.2.2 Facade Parsing

For street scenes, classical image segmentation techniques [157] have been extended with architectural scene segmentation using color and contour features [12]. Additional sources of information such as a height prior [206, 207, 23] or object detectors [144, 113] are typically introduced on top of local features. However, classification is performed mainly on 2D images, whereas 3D is introduced only at a procedural level [182, 162, 125].

To capture the structure inherent to facades, different approaches have been proposed. Several utilize shape grammars to learn and exploit the structure in facades.

Alegre and Dellaert [3] model facades with stochastic context-free grammars and rjMCMC sampling. Müller *et al.* [126] use regular grids to infer procedural CGA grammar for repetitive facades. Shen *et al.* [154] assume multiple interlaced grids and provide a hierarchical decomposition. A similar assumption of block-wise decompositions can be used to parse facades using a binary split grammar [208]. Simon *et al.* [161] use a specialized Haussmannian facade grammar coupled with an optimization based on random walks, and Reinforcement Learning [181]. Simon *et al.* [162] additionally use 3D depth information in a GA optimization framework. Cohen *et al.* [30] propose efficient DP subproblems which hard-code the structural constraints. Moving further away from hard-coded shape grammars, Riemenschneider *et al.* [144] use irregular lattices to reduce the dimensionality of the parsing problem, modeling symmetries and repetitions. Kozinski *et al.* [97] relaxes the Haussmannian grammar to a graph grammar where structure and position are optimized separately.

Moving entirely away from strict grammars, Dai *et al.* [35] use a facade-specific segmentation together with learning weights for different meta-features capturing the structure. Tyleček *et al.* [192] model alignment and repetition through spatial relations in a CRF framework.

In Chapter 5, we proposed a three-layered approach introducing 2D weak architectural principles instead of rigid shape grammar rules. Since our goal is still to impose as few restrictions on the facade structure as possible, we build upon this work by introducing 3D architectural principles and an elegant optimization formulation based on integer programming. This allows us to obtain better quality of the reconstructed facades with significant speed improvement over their 2D counterparts.

6.2.3 Facade Separation

All of the aforementioned works on facade parsing assume individual facades to be separated beforehand. Yet, this separation is not trivial by far and quite a few automated pipelines gloss over the issue.

In the full 3D domain, most work focuses on building extraction, which deals with 2.5D height models and identifies individual building blocks and their roof types based on height information [104].

Similar approaches have been adopted for street-level data, where height is rightfully used as the most discriminative feature [207, 216]. Other methods deal with repetitive features that reoccur on one facade and not on neighboring facades [199]. These work well if the assumptions are correct, i.e. the buildings have different heights and a quite different appearance of their facades. However, some architectural styles aim at similar appearances and heights. Furthermore, methods working in 2D require the facades to be completely visible in each image, such as our edge-based facade separation approach in Chapter 3.

As an alternative to these approaches, we propose a semantic facade separation approach. The aim of this approach is to exploit the usually varying layout of semantic structures in different facades, which may share the same style. In contrast to methods for facade structure understanding [198, 205] which require already split facades, we propose to use the semantic scene knowledge to create these splits.

6.3 3D Semantic Facade Segmentation

Our goal is to estimate a semantically segmented 3D scene starting from images of an urban environment as input. As a first step, we obtain a set of semi-dense 3D points from standard SfM/MVS algorithms [204, 55, 83].

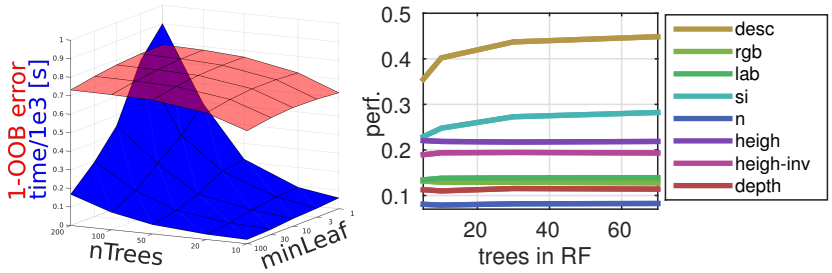


Figure 6.2: **Parameters of the 3D classifier.** Left: performance (red) and test time (blue) w.r.t. the number of trees and the minimum number of observations per leaf. Right: the performance of each descriptor part individually (and the final descriptor-desc) using RF classifier.

Next, we classify each point P_i in the point cloud into one semantic class L_i (window, wall, balcony, door, roof, sky, shop), using a Random Forest classifier trained on light-weight 3D features (Sec. 6.3.1). Afterwards, we separate individual facades by detecting differences in their semantic structure (Sec. 6.3.2). Finally, we propose architectural rules that express preferences such as the alignment or co-occurrence of facade elements. These rules have two effects: they improve our results and directly return the high-level 3D facade structure (Sec. 6.3.3).

6.3.1 Facade Labeling

We create the initial labeling of the 3D scene by employing a Random Forest (RF) classifier on the following set of descriptors for each 3D point P_i: mean **RGB** colors of the point as seen in the camera images; the **LAB** values of that mean RGB [143]; normal (**n**) at the 3D point; 3D geometry captured using the spin-image (**SI**) descriptor [84], calculated on different scales; the point's height (**h**) above the estimated ground plane; its "inverse height" ($\mathbf{h}^{-1}$), defined as the distance from the uppermost point of the facade in the direction of the gravity vector; depth (**dph**) defined as the distance of the point P_i to the approximate facade plane. Since we do not have the facade separation available yet, we estimate $\mathbf{h}^{-1}$ and **dph** from the subset of 3D points assigned to their nearest camera. Thus, the full descriptor per point P_i is:

$$\underset{132\times1}{\mathbf{d}_i} = [\underset{3\times1}{\mathbf{RGB}_i^\mathsf{T}}\ \underset{3\times1}{\mathbf{LAB}_i^\mathsf{T}}\ \underset{3\times1}{\mathbf{n}_i^\mathsf{T}}\ \underset{120\times1}{\mathbf{SI}_i^\mathsf{T}}\ \underset{1\times1}{\mathbf{h}_i^\mathsf{T}}\ \underset{1\times1}{\mathbf{h^{-1}}_i^\mathsf{T}}\ \underset{1\times1}{\mathbf{dph}_i^\mathsf{T}}]^\mathsf{T}.$$

Once this 132-dimensional descriptor is known for each point P_i, we train an RF classifier with a uniform class prior. All classification parameters, such as

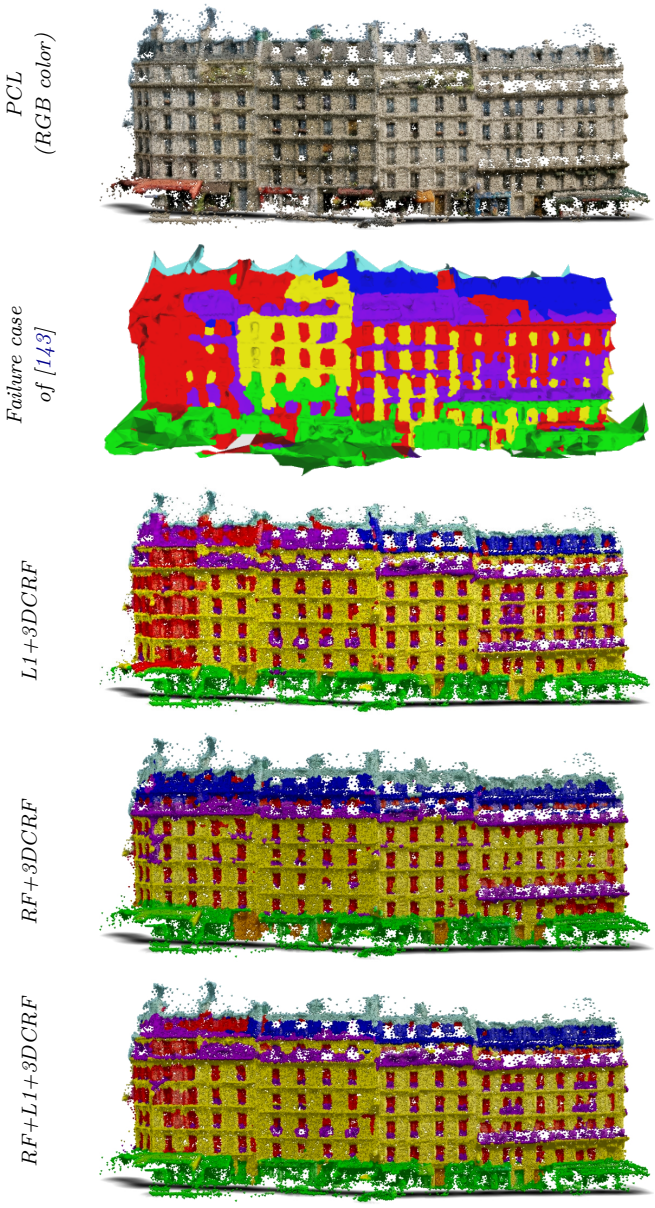


Figure 6.3: **Qualitative results of the facade labeling step.** A challenging subset of RueMonge2014, dubbed Sub28 in [143]. Interestingly, the 2D and 3D-based methods (third vs. fourth row) outperform each other for different parts of the scene, while their combination (fifth row) has the best performance.

scales of the SI descriptor $(0.15, 0.3, 0.45)$, number of trees (100) and minimum leaf size (30) in the RF are determined using grid search on out-of-bag (OOB) error estimates. The effect of these parameters and the impact of each utilized 3D descriptor on classifier performance are shown in Figure 6.2.

6.3.2 Facade Splitting

Given the point cloud $P = \{P_i\}$ and its labeling results $L = \{L_i\}$ with the best class label L_i assigned to each individual 3D point P_i, a novel method for separating individual facades is proposed. The underlying issue with previous work is that typical features such as height or appearance are too weak, especially in strongly regulated urban scenes, such as Haussmannian architecture in Paris.

In this work we propose a facade separation method that exploits semantic nuances between facades. Despite the strong similarity of buildings, even in Haussmannian style, each facade shows individual characteristics such as window heights, balcony placements and roof lines. This knowledge is only available after performing semantic classification.

In order to separate facades into individual units, we vertically split the dominant facade plane, by posing a labeling problem that assigns sites $S = \{s_i\}$ (single connected components within the classified point cloud P) to facade groups $G = \{g_i\}$ is defined as:

$$E(S) = \sum \Theta(g_i, s_i) + \lambda \cdot \sum \Psi(s_i, s_j) \tag{6.1}$$

where $\Theta(g_i, s_i)$ determines the cost of assigning a site s_i to a facade group g_i, equal to its distance in 1D location. The pairwise term $\Psi(s_i, s_j)$ encourages splits where there is decreased likelihood of crossing any architectural elements, such as windows or balconies. It aggregates the class labels in vertical direction and estimates a ratio between wall class (where the split should occur) and structural classes (such as windows, balconies, etc. where no split should be selected).

Each facade group g_i is a label which defines a candidate layout for an individual facade. It is determined by clustering features F capturing the differences between semantic elements. These features are statistical measurements defined as

$$\mathbf{F_i} = [\delta_i^\mathsf{T},\ \mathbf{A}_i^\mathsf{T},\ \mathbf{Major}_i^\mathsf{T},\ \mathbf{Minor}_i^\mathsf{T},\ \mathbf{verthist}(C_i)] \tag{6.2}$$

where for each connected component, δ is the position along the dominant plane, $\mathbf{A}$ is its area, **Major**, **Minor** are the lengths of its axes and **verthist** is

the histogram over the class labels above and below this connected component. These features are clustered using Affinity Propagation [52] and determine the facade groups G.

The final assignment is optimized with a multi-label graphcut [21, 20, 92], which assigns all original 3D points P with respect to their distance to one of the facade groups G, and the final labeling determines the number of facades.

For intermediate evaluation, we manually annotate the splits between individual facades, and evaluate how many facades were correctly separated. The achieved accuracy in terms of correct facade-wise classification is 95.5% where all but 3 facades have at least 97% 3D points assigned correctly. A baseline using F=RGB+intensity gives only 78% overall, failing to split four and oversplitting additional five.

6.3.3 Weak Architectural Principles (WP)

In order to generate a more structured final output, and building upon the 2-dimensional weak architectural principles introduced in Chapter 5, we use generic principles such as *alignment*, *symmetry*, *co-occurrence*, and *vertical region order*. The main idea of this approach is that some principles are used to discover the candidate objects in the facade, while others score the elements or modify their position and size.

3D Principles (3DWP)

We propose a generalization of the 2D architectural principles to 3D with several major differences. Unlike Chapter 5, where initial elements are determined with a simple connected component analysis on a binary 2D image, we discover them in a robust way directly in the point cloud. Second, since our approach works with 3D boxes instead of bounding rectangles, our approach implicitly models the z-position (along the facade normal) and depth of each facade element. Furthermore, we generalize the alignment rule to synchronize same-class elements in the z-direction. This allows us to naturally model facade configurations with inset windows or extruding balconies. As will be demonstrated later in the text, this particular choice of 3D principles allows us to use a simple and fast optimization framework.

Our goal is to find the optimal set of boxes ($\mathcal{B}$) which (1) fits well to the initial data labeling (L); (2) has well *aligned* boxes; (3) does not contain overlapping elements of the same class; (4) satisfies element *co-occurrence*, e.g. a balcony should not appear if there is no window above it. We formulate this optimization

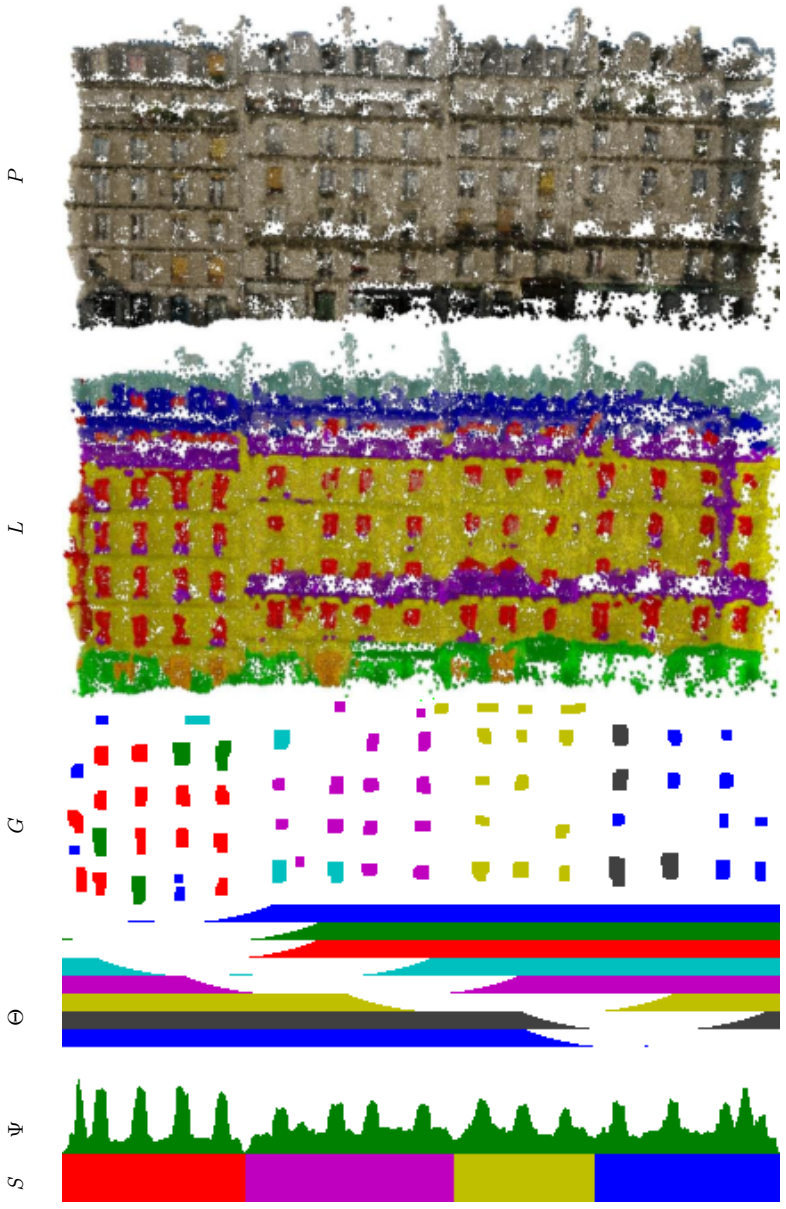


Figure 6.4: Exemplar facade split projected into 2D for visualization (top to bottom): 3D colored points, 3D classification, group prototypes (here windows), unary/pairwise costs and final 1D group assignment. Note the high similarity in appearance and height.

as follows:

$$\begin{aligned} &\arg\min_{\mathcal{B}\in\mathcal{B}^{\text{super}}} \big(f_{\text{data}}(\mathcal{B}, P, L) + f_{\text{align}}(\mathcal{B}) \big). \\ &\quad s.t. \quad c_{\text{overlap}}(\mathcal{B}) = 0 \\ &\qquad\quad c_{\text{co-occ}}(\mathcal{B}) = 0 \end{aligned} \tag{6.3}$$

Generating the initial set of boxes. From our initial point cloud labeling L, we generate an over-complete set of boxes $\mathcal{B}^{\text{super}}$. Note that in Sec. 5.6.1, the initial elements are generated by finding connected components in a labeled 2D image, followed by fitting of minimal bounding rectangles. Performing the similar task in 3D raises two main issues. First, we cannot use the 4- or 8-connected neighborhood to discover the connected components, as we deal with 3D points in continuous space. Second, the 2D approach often generates too large elements, e.g. in presence of significant noise, when distinct facade elements appear connected in L.

In the 3D case, for each object class c (window, balcony, door) we create a binary labeling L^c, where $L^c_i = 1$ if $L_i = c$ and 0 otherwise. We extract the initial facade elements from the labeling L^c by creating a K-nearest neighbor graph in 3D ($K = 7$ in our experiments), and discarding edges that connect nodes labeled with 1 and 0. We fit a 3D box to each component, and add it to $\mathcal{B}^{\text{super}}$.

However, since the labeling L^c can be quite noisy, we clean it up with the generalization of the morphological *opening* (erosion followed by dilation) operator to 3D. The erosion operator changes the label of a point to 0 if any of its K nearest neighbors is labeled with 0, while the dilation performs the opposite process. By varying the number of subsequent erosions and dilations, we generate multiple overlapping proposals for each facade element, with different degrees of smoothing – all used to augment $\mathcal{B}^{\text{super}}$.

Finally, we use the *symmetry* principle to add elements which are potentially missing in the scene. We detect the main vertical symmetry plane of the facade, mirror all elements and add them to $\mathcal{B}^{\text{super}}$.

Best set of boxes. We pose the search problem in Eq. 6.3 as an integer quadratic program (IQP) with linear constraints. Each box $\mathcal{B}_i \in \mathcal{B}$ is assigned an indicator variable $\mathbf{x_i} \in \mathbf{x}$, which is equal to 1 if the box is selected in the set,

0 otherwise. The IQP is formulated as follows:

$$\begin{aligned} \min \quad & \mathbf{w}_{\text{data}}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{Q}_{\text{align}} \mathbf{x} \\ s.t. \quad & \mathbf{C}_{\text{overlap}} \mathbf{x} \leq \mathbf{1} \\ & \mathbf{C}_{\text{co-occ}} \mathbf{x} \geq \mathbf{0} \\ & \mathbf{x}_i \in \{0, 1\} \end{aligned} \tag{6.4}$$

For each box B_i with label c, we set the *data* term $\mathbf{w}_{\text{data}}(i) = |L(\mathcal{B}_i) = c| - |L(\mathcal{B}_i) \neq c|$, and then normalize $\mathbf{w}_{\text{data}}$ to sum up to unity.

The *alignment* term is defined for pairs of boxes $\mathcal{B}_i$ and $\mathcal{B}_j$. We distinguish 6 types of alignment: top and bottom, left and right, back and front. For each type, two boxes are aligned if the corresponding edges of the boxes are within a threshold. We set this threshold equal to half the median size of objects of the same class, in the appropriate direction.

$$\mathbf{Q}_{\text{align}}(i, j) = \begin{cases} -a & \text{if } \mathcal{B}_i \text{ and } \mathcal{B}_j \text{ are aligned } a \text{ times} \\ 0 & \text{otherwise.} \end{cases} \tag{6.5}$$

To make sure that the resulting quadratic program is convex, we make the resulting matrix diagonally dominant, and therefore positive semi-definite:

$$\mathbf{Q}_{\text{align}}(i, i) = \sum_{j, j \neq i} |\mathbf{Q}_{\text{align}}(i, j)| \tag{6.6}$$

Every row of the *overlap* constraint matrix $\mathbf{C}_{\text{overlap}}$ ensures that a pair of same-class overlapping boxes (IOU>0) $\mathcal{B}_i$ and $\mathcal{B}_j$ cannot be selected at the same time:

$$\mathbf{x}_i + \mathbf{x}_j \leq 1 \tag{6.7}$$

The *co-occurrence* principle prohibits balconies without at least one window on top:

$$\mathbf{C}_{\text{co-occ}}(i, j) = \begin{cases} -1 & \text{if } i = j \text{ and } \mathcal{B}_i \text{ is a balcony.} \\ 1 & \text{if } \mathcal{B}_i \text{ is a balcony and} \\ & \mathcal{B}_j \text{ is an adjacent window.} \\ 0 & \text{otherwise.} \end{cases} \tag{6.8}$$

The optimization in Eq. 6.4 is solved using the MOSEK mixed-integer solver via the CVX software package [68].

Figure 6.5: **Estimated 3D facades.** All reconstructed facades in the RueMonge2014 test set. Our method performs automatic separation of facades and analyzes the 3D structure of facade elements. The final results are obtained by fitting 3D boxes to the discovered objects and texturing with ortho-images. Please zoom in to view 3D structure, or consult the detailed view in Fig. 6.8.

Locations and sizes of boxes. The optimization of Eq. 6.3 does not modify the size or location of the selected boxes. We generalize the alignment principle from Sec. 5.6.2 to perform alignment in three main directions of the facade. The objective function is defined as the sum of Tukey's bounded influence functions [215] evaluated on absolute differences of bounding box edge positions, for all pairs of boxes. We solve for the box locations and sizes using a Quasi-Newton optimization approach. In essence, this optimization "snaps" the borders of nearly-aligned elements to common alignment lines. Unlike its 2D counterpart, this process allows the creation of depth-aligned windows and balconies.

Ortho + 2D Principles (2DWP)

As mentioned earlier, the method presented in Sec. 5.6 requires rectified 2D images and labelings as input to its third layer. In order to use it in our 3D scenario, we create a "virtual 2D" input for each facade. We start by a least-square plane fit to the 3D points of the facade. The points P and their labels L are then projected onto the plane. The ortho labeling is generated by uniformly sampling points on this plane, and finding the nearest projected point for each pixel. The downside of this method is that useful 3D information is lost in the process. Furthermore, the processing time is increased due to the overhead of 2D-3D projections.

6.4 Evaluation

We consider three tasks pertaining to facade labeling: *point cloud labeling*, *image labeling*, and *facade parsing*. In all experiments, to evaluate the semantic

segmentation results, we use the PASCAL-VOC IoU segmentation accuracy, averaged per class. Qualitative results on point clouds are shown by overlaying the labeling on the colored point cloud, see Fig. 6.8.

Datasets. We perform exhaustive evaluations on the only publicly available, street-side facade dataset RueMonge2014[2] introduced by Riemenschneider *et al.* [143], which contains a 700-meter-long street in Paris. This is of particular interest for two main reasons. First, the facade separation problem is a very difficult one due to the homogeneous architectural appearance. Second, to best of our knowledge, none of the other datasets [23, 59, 103, 150] provide such dense labels in 3D. As we focus on point cloud labeling, we consider only the vertices from RueMonge2014 mesh, which we name 'Low-res', since it was generated by 2.7x subsampling the original, 'High-res' mesh produced by the CMPMVS algorithm [83] in 270 minutes. For reconstruction speedup, one could use methods from Bodis-Szomoru *et al.* [14, 15] who densely reconstruct the scene on a single core in roughly two seconds/image (14 min), or the commercial version of the CMPMVS algorithm [24] which reconstructs the same scene on the GPU in only 4 minutes. For completeness, we evaluate our 3D classifier on two additional point clouds: sparse 'SfM' (using VisualSFM [203], 13 min) and semi-dense 'PMVS' (using [55], 21 min).

6.4.1 Point Cloud Labeling

We compare several approaches for 3D point cloud labeling, see Table 6.1 and Fig. 6.3. First, as the example of a purely-3D approach, we use our initial Random Forest classifier (RF+MAP). The result is then smoothed with a 3D Conditional Random Field (RF+3D CRF). The Potts model-based pairwise potentials are defined over a 4-nearest neighbor graph of the point cloud.

This result is compared with state-of-the-art 2D facade labeling introduced in Chapter 5. Its first two layers will be referred to in this section as L1 and L2, respectively. The resulting semantic segmentation of images is projected and aggregated in the point cloud by either majority voting from different cameras, or using the aforementioned 3D CRF.

Finally, we combine the results of 3D and 2D methods using the CRF, resulting in a higher performance at the cost of evaluation time. We compare these hybrid methods to the recent approach that combines 2D and 3D for facade labeling [143], and observe significant improvement in quality. It is worth noting that the joint 2D+3D approach gives the best performance but at a 26× lower

[2] http://varcity.eu/3dchallenge/

Point cloud labeling		Low-res PCL		High-res PCL	
Method		Accuracy	Timing	Accuracy	Timing
3D	RF+MAP	51.42	15min	55.65	76min
	RF+3D CRF	**52.09**		**56.39**	
2D	L1+majority vote	54.68	302min	53.37	305min
	L1+MAP	55.35		54.06	
	L1+3D CRF	55.72		54.30	
	L2+majority vote	56.10	382min	54.74	385min
	L2+MAP	55.95		54.71	
	L2+3D CRF	**56.32**		**54.95**	
3D+2D	[143]	42.32	15min	39.92	23min
	RF+L1+MAP	60.16	317min	61.15	381min
	RF+L1+3D CRF	60.05		61.21	
	RF+L2+MAP	**60.44**	397min	61.31	461min
	RF+L2+3D CRF	60.43		**61.39**	

Table 6.1: Semantic segmentation of point clouds: accuracy for various methods on the RueMonge2014 dataset. We report the results on the low- and high-resolution point clouds as PASCAL IOU accuracy in %. The evaluation time includes feature extraction, classification, and optional 3D projection.

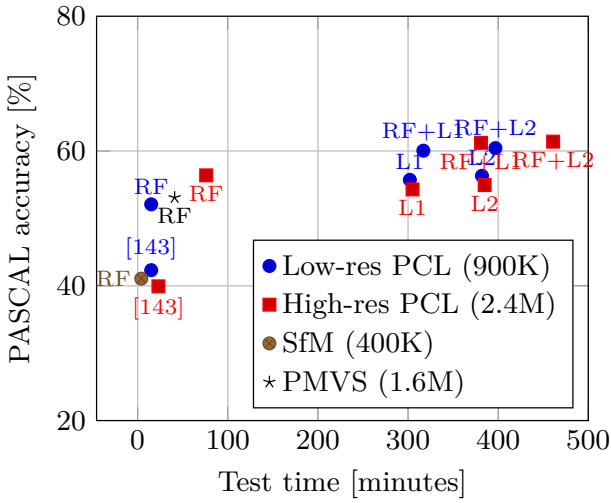


Figure 6.6: PCL labeling: accuracy vs. test time for two different point cloud resolutions generated by CMP. We also show the performance of our RF method on the point clouds generated by SfM and PMVS. The number of vertices per cloud is shown in parentheses.

Image labeling		Low-res PCL		High-res PCL	
Method		Accuracy	Timing	Accuracy	Timing
3D	RF+MAP	52.85	21min	57.82	85min
	RF+3D CRF	**53.22**		**58.13**	
2D	L1	54.46	299min	54.46	299min
	L2	**57.53**	379min	**57.53**	379min
3D+2D	[143]	41.34	15min	n/a	n/a
	RF+L1+MAP	61.58	324min	63.08	390min
	RF+L1+3D CRF	61.27		62.87	
	RF+L2+MAP	**61.95**	404min	**63.32**	470min
	RF+L2+3D CRF	61.73		63.13	

Table 6.2: Semantic segmentation of street-side images: accuracy for various methods on the RueMonge2014 dataset. The results are shown for the test set of 202 test images. The 2D results are obtained by running the first two layers (L1 and L2) of the 3Layer method [113], and projecting the point cloud classification onto the original images. The PASCAL IOU accuracy is shown in % over the image domain.

speed and with a modest 8% accuracy gain over the 3D-only approach. The high-res point cloud increases the performance of the 3D-only classifier by 4% but at the cost of a 5× lower speed.

6.4.2 Image Parsing

Comparison to 2D methods is additionally performed in the image domain, by back-projecting our point cloud labeling L onto the perspective images, and filling out gaps with nearest-neighbor interpolation, see Table 6.2. In the image domain, a similar behavior is observed, as the 3D-only approach achieves the highest speed and competitive results to the 2D-only classification, which is only 4% better but 18× slower. The complementary combination of 2D and 3D again achieves top performance (63.32%), outperforming the existing method [143] by over 20%.

6.4.3 Facade Parsing

We compare the proposed 3D version of the weak architectural principles (3DWP) with its 2D counterpart (2DWP) from Chapter 5, see Table 6.3. The evaluation is performed in the original point cloud, by concatenating the individual facade labelings (3D) or back-projecting the labeled ortho-images (2D).

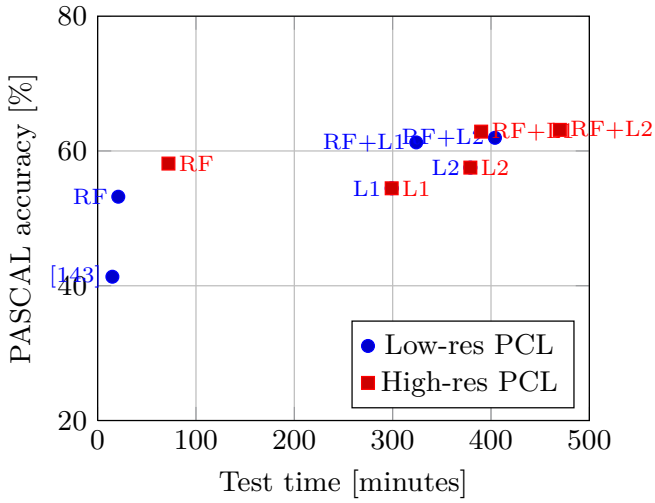


Figure 6.7: Image labeling: accuracy vs. test time for two different PCL resolutions.

Facade parsing		Low-res PCL		High-res PCL	
Method	Input classification	Accuracy	Timing	Accuracy	Timing
2DWP	3D: RF+3D CRF	49.59	802min	49.54	885min
	2D: L1+3D CRF	54.04		53.29	
	3D+2D: RF+L1+3D CRF	**58.81**		**58.40**	
3DWP	3D: RF+3D CRF	52.24	8min	56.35	10min
	2D: L1+3D CRF	55.39		53.56	
	3D+2D: RF+L1+3D CRF	**60.83**		**59.89**	

Table 6.3: Semantic segmentation of street-side point clouds using weak architectural principles (WP) on the RueMonge2014 dataset. We compare the original 2D version applied on virtual ortho-images, and our proposed 3D method, for the three representative classifiers from Table 6.1. The PASCAL IOU accuracy is shown in %. The test time does not include the time needed for the initial point cloud classification.

We test three different classifiers as input to this stage, based on features from 3D, 2D and 2D+3D. Our 3DWP approach outperforms its 2D counterpart in all cases except when using the 2D-only input. However, the most obvious improvement is the speed of our IQP optimization compared to the GA-based approach in Sec. 5.6.1.

Overall, top performance is achieved by a combination of 2D and 3D features and pure 3D weak architectural principles in 325 minutes (317 for initial labeling + 8 weak principles). The fastest, yet still competitive performance uses only 3D features and 3D weak principles, which requires roughly 20 minutes from start (point cloud) to end (textured 3D models) for the full street.

A visual comparison of the stages is shown in Fig. 6.8, including the original color point cloud, initial classification, the result of 3D weak architectural principles using the best classifier, and final geometry-correct textured models. For an overview of all facades reconstructed in 3D, see Fig. 6.5.

6.5 Conclusion

This chapter proposed a new approach for 3D city modeling using 3D semantic classification, 3D facade splitting, and 3D weak architectural principles. The proposed method produces state-of-the-art results in terms of accuracy and computation time. The results indicate that 3D-only classification is feasible and leads to significant speedups.

Overall, the benefit of processing all stages in 3D enables us to better exploit the single domain and advantages of combined depth information. A possible extension to this system would be to use feedback to the original SfM point cloud creation, in order to join the two processes.

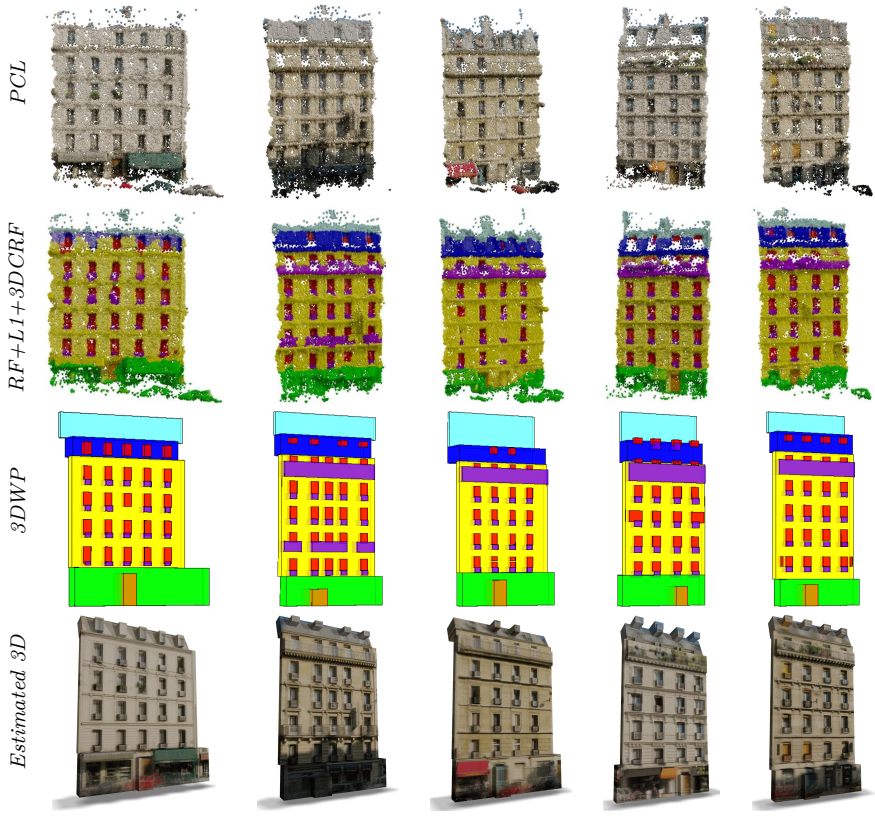


Figure 6.8: **Qualitative results.** We show examples of automatically obtained facades using our method. From top to bottom: initial colored point cloud (Low-res), initial classification, estimated boxes using weak 3D principles; and –as we suggested that 3D semantic interpretation can be used to estimate 3D shape– automatically generated 3D model of the facades textured by projecting ortho images.

Part III

Grammar Learning

Chapter 7

Bayesian Grammar Learning

So far, we have demonstrated the usability of procedural grammars for building reconstruction (Chapter 4), and ways of obtaining reasonable bottom-up reconstructions in the absence of pre-written grammars (Chapters 5 and 6). Now we focus our attention on the core problem, which is inferring these grammars from data at hand. Traditional grammar-based approaches are limited in scope, as they require a human expert to write grammars for each encountered building style. Instead, we propose to learn these grammars from data. The approach presented in this chapter[1] allows us to automatically learn a special kind of procedural grammars from a set of labeled building facades. Namely, we learn two-dimensional attributed stochastic context-free grammars (2D-ASCFGs). The basic technique used to achieve this is Bayesian Model Merging, originally developed in the field of natural language processing, which we extend to the domain of two-dimensional languages. Given a set of ground-truth labeled facade images, we induce a grammar which can be sampled to create novel instances of the same building style. In addition, we demonstrate that our learned grammar can be used for parsing existing facade imagery. Experiments conducted on the dataset of Haussmannian buildings in Paris show that our parsing with learned grammars not only outperforms bottom-up classifiers but is also on par with approaches that use a manually designed style grammar.

[1] This chapter is based on the joint work with Luc Van Gool, published in CVPR 2013 [114]. Anđelo Martinović is the first author.

Figure 7.1: "Somewhere in Paris": a street with buildings sampled from our induced grammar.

7.1 Introduction

Many existing approaches use some form of shape grammars as higher-order knowledge models for reconstruction of buildings. Vanegas *et al.* [195] used a simple grammar for buildings that follow the Manhattan world assumption. A grammar was fitted from laser-scan data in [189]. An approach using reversible jump Markov Chain Monte Carlo (rjMCMC) for fitting split grammars to data was described in [145]. Teboul *et al.* [180] presented an efficient parsing scheme for Haussmannian shape grammars using Reinforcement Learning. We have also proposed in Chapter 4 an approach that combines shape grammars with object detectors in order to faithfully reconstruct Greek Doric temples.

However, all of the methods mentioned above share a common drawback. They assume that a manually designed grammar is available from the outset. This is a serious constraint, as it limits the reconstruction techniques to a handful of building styles for which pre-written grammars exist. Creating style-specific grammars is a tedious and time-consuming process, which is usually performed only by a few experts in the field. So, a natural question arises: can we learn procedural grammars from data?

So far, the research in the field of general IPM has been limited to a small number of approaches. Learning L-systems from synthetic 2D vector data was

tackled in [166]. Applications of general IPM in urban modelling started with Aliaga *et al.* [4], who presented an interactive method for extracting facade patterns from images. Bokeloh *et al.* [17] learned deterministic shape grammar rules from triangle meshes and point clouds. Attribute graph grammars [74] were presented as a method of top-down/bottom-up image parsing, though restricting the detected objects in scenes to rectangles.

In the field of formal grammar learning, a famous conclusion of Gold [64] states that no superfinite family of deterministic languages (including regular and context-free languages) can be identified in the limit. However, Horning [79] showed that the picture is not so grim for statistical grammar learning, and demonstrated that stochastic context-free grammars (SCFGs) can be learned from positive examples. Currently, one of the popular methods for learning SCFGs from data is Bayesian Model Merging [170], which makes the grammar induction problem tractable by introducing a Minimum Description Length (MDL) prior on the grammar structure. This approach was recently applied for learning probabilistic programs [82] and design patterns [176].

7.2 Overview

Inspired by the aforementioned success stories of Bayesian Model Merging outside computer vision, we propose a novel approach of inducing procedural models, particularly *split grammars*, from a set of labeled images. We focus our discussion on facade modeling, since facades are mostly two-dimensional, and exhibit a logical hierarchy of elements.

The overview of our grammar learning approach can be seen in Fig. 7.2. The input to our system is a set of facade images, which are semantically segmented into classes such as walls, windows, etc. In the first step we create a stochastic grammar which generates only the input examples with equal probabilities. However, we want to find a grammar that can also generalize to create novel designs. We formulate this problem as a search in the space of grammars, where the quality of a grammar is defined by its posterior probability given the data. As described in Sec. 7.5, this requires an optimal trade-off between the grammar description length (smaller grammars are preferred) and the likelihood of the input data. The latter is obtained by parsing the input examples with the candidate grammar.

Previous work has shown that image parsing with a known set of grammar rules is a difficult problem by itself [144, 182]. On the other hand, our grammar search procedure will need to evaluate a huge number of candidate grammars in order to find the best one, as will be shown further in the text. This means

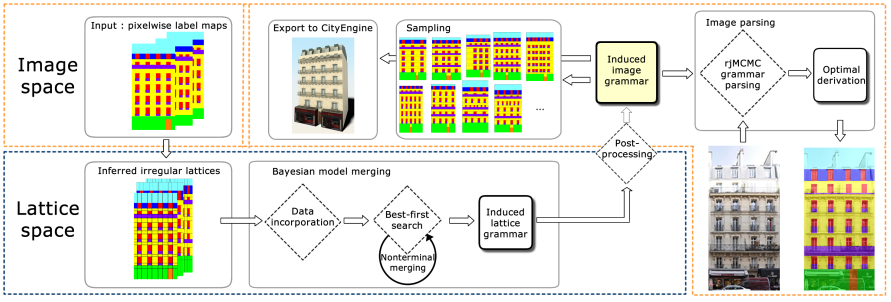


Figure 7.2: Overview of our Bayesian grammar learning approach.

that we have to be able to parse the input examples in a very short time, lest the grammar search last indefinitely. Different authors have tackled this *curse of dimensionality* during parsing in different ways: assuming that the building exhibits a highly regular structure [126], using approximate inference such as MCMC [145], or exploiting grammar factorization [182]. The work by Riemenschneider *et al.* [144] has shown that it is possible to perform exact image parsing using dynamic programming if the image is reduced to an irregular lattice (a grid structure). This approach reduces the effective dimensionality of the problem, while not sacrificing much of the quality.

Following their example, we transform all of our input images into irregular lattices, casting our grammar search procedure into a lower-dimensional space. In this space we use our own, modified version of the Earley-Stolcke parser [170], a technique from natural language processing adapted to parse 2D lattices instead of 1D strings. This dimensionality reduction enables the grammar search procedure to run within a reasonable time. Finally, in order to perform image parsing, the induced grammar is cast into the original space. The resulting stochastic, parameterized grammar can either be used as a graphics tool for sampling building designs, or as a vision tool to alleviate image parsing of actual buildings.

The contributions of this chapter are:

1. A novel approach for inducing procedural split grammars from data. To the best of our knowledge, we are the first to present a principled approach for learning probabilistic two-dimensional split grammars from labeled images.

2. A generalization of the Earley-Stolcke SCFG parser to two dimensional lattices.

3. An adapted rjMCMC parser in the style of Talton *et al.* [175] for image-scale parsing.

4. An experimental evaluation suggesting that learned grammars can be as effective as human-written grammars for the task of facade parsing.

7.3 2D-ASCFGs

We define a two-dimensional attributed stochastic context-free grammar (2D-ASCFG) as a tuple $G = (N, T, S, R, P, A)$, where N is a set of non-terminal symbols, T a set of terminal symbols, S the starting non-terminal symbol or axiom, R a set of production rules, $\{P(r), r \in R\}$ a set of rule probabilities and $\{A(r), r \in R\}$ a set of rule attributes.

Every symbol is associated with the corresponding shape, representing a rectangular region. Starting from the axiom, production rules subdivide the starting shape either in horizontal or vertical directions. We define the set R as a union of horizontal and vertical productions: $R = R_h \cup R_v$. These productions correspond to standard horizontal and vertical split operations in split grammars (see Sec. 2.1.4). A production is of the form $X \to \lambda$, where $X \in N$ is called the left-hand-side (LHS), and $\lambda \in (N \cup T)^+$ is called the right-hand-side (RHS) of the production.

For every production we define $P(X \to \lambda)$ as the probability that the rule is selected in the top-down derivation from the grammar. For the grammar to be well-formed, the productions with X as LHS must satisfy the condition $\sum_\lambda P(X \to \lambda) = 1$. We additionally associate each grammar rule r with a set of attributes $A(r) = \{\alpha_i\}$. The elements of a single attribute are the relative sizes of the RHS shapes in respect to their parent shape, in the splitting direction: $\alpha_i = \{s_1, ..., s_{|\lambda|}\}, \sum_i s_i = 1$. These relative sizes sum up to unity because RHS shapes always fill the entire shape of their parent.

We denote by τ a parse tree from the grammar, rooted on the axiom, its interior nodes corresponding to non-terminal symbols, and its exterior nodes to terminal symbols. The parse tree is obtained by applying a sequence of rules on the axiom and non-terminal nodes. A derivation from the grammar consists of the parse tree and the selected attributes at each node: $\delta = (\tau, \alpha)$. The probability of a single derivation is the product of all rule probabilities selected at each node s of the parse tree: $P(\delta) = \prod_{s \in \delta} P(r_s)$. The set of terminal nodes of a parse tree defines a lattice over an area. A lattice is a rectangular tesselation of 2D space, exactly filling the shape of the axiom. We define the likelihood of the

grammar G generating a lattice l as $L(l|G) = \sum_{\delta \Rightarrow l} P(\delta)$, where we sum over the probabilities of all derivations that yield a particular lattice.

7.4 Bayesian Model Merging

To cast our grammar learning problem as an instance of Bayesian Model Merging, we need to define several methods:

- **Data incorporation**: given a body of data, build an initial grammar which generates only the input examples.
- **Model merging**: propose a candidate grammar by altering the structure of the currently best grammar.
- **Model evaluation**: evaluate the fitness of the candidate grammar compared to the currently best grammar.
- **Search**: use model merging to explore the grammar space, searching for the optimal grammar.

7.4.1 Data Incorporation

We start with a set of n_f facade images, with each pixel labeled as one of the n_l terminal classes (window, wall, balcony, etc.) As already mentioned in Sec. 7.1, grammar induction would be infeasible in the image space due to the curse of dimensionality. To mitigate this issue, all input images are converted into lattices following an approach similar to Riemenschneider *et al.* [144]. Every rectangular region in the resulting two-dimensional tesselation of the image is labeled with the majority vote from the corresponding pixel labels.

For each lattice in the input set, we create an instance-specific split grammar, with terminal symbols corresponding to image labels. Non-terminal productions are created by alternatively splitting the image in horizontal and vertical directions, starting with the latter. All production probabilities are set to 1; all attributes are initialized to the relative sizes of right-hand side elements. For example, the first production splits the axiom into horizontal regions represented by newly instantiated non-terminals and parametrized by their height: $S \rightarrow X_i \dots X_n, p = 1, A = \{\{h(X_i), \dots, h(X_n)\}\}$, where the rule probability p is initialized to 1, but is allowed to change in the model search. The procedure is stopped at the level of a single lattice element, where we instantiate lexical productions, i.e. productions with a single terminal on the

RHS: $X \to label, p = 1, A = \{\{1\}\}$. Lexical productions remain deterministic, as they only label the entire shape of the parent with the given terminal class.

Now we have a set of deterministic grammars G_i, each producing exactly one input lattice. The next step is to merge them into a single grammar by setting all of their axioms to the same symbol and aggregating all symbols and productions: $G_0 = (\cup N_i, \cup T_i, S, \cup R_i, \cup P_i, \cup A_i)$. The probabilities of the rules starting from the axiom are changed to $1/n_f$, which means that the grammar G_0 generates each of the input examples with the same probability.

7.4.2 Merging

A new grammar is proposed by selecting two non-terminals X_1 and X_2 from the current grammar and replacing them with a new non-terminal Y. This operation has two effects on the grammar. First, all the RHS occurrences of X_1 and X_2 are replaced by Y:

$$Z_1 \to \mu_1 X_1 \lambda_1 \qquad \underset{\dashrightarrow}{merge} \qquad Z_1 \to \mu_1 Y \lambda_1$$

$$Z_2 \to \mu_2 X_2 \lambda_2 \qquad\qquad\quad Z_2 \to \mu_2 Y \lambda_2$$

where $\mu, \lambda \in (N \cup T)^+$. If $Z_1 = Z_2, \mu_1 = \mu_2, \lambda_1 = \lambda_2$, then the two resulting productions are merged in one. In that case, the attribute set of the new production is defined as the union of the attributes of the old productions.

Second, all the productions where X_1 and X_2 appear on the LHS are replaced with Y, as well:

$$X_1 \to \lambda_1 \qquad \underset{\dashrightarrow}{merge} \qquad Y \to \lambda_1$$

$$X_2 \to \lambda_2 \qquad\qquad\quad Y \to \lambda_2$$

Again, if $\lambda_1 = \lambda_2$, only one production is created. If we create a production $Y \to Y$, we delete it from the grammar.

The merging operation basically states that in the resulting grammar two previously different symbols may be used interchangeably, although with different probabilities. The only restriction that we place on the merging operations is that X_1 and X_2 have to be "label-compatible", meaning that the sets of terminal symbols reachable from both nodes have to be equal. In this way we prevent nonsensical merges, e.g. merging two non-terminals representing sky and door regions, respectively. We also improve the speed of the inference procedure by restricting the search space.

7.4.3 Evaluating Candidate Grammars

The goal is now to find the grammar model G that yields the best trade-off between the fit to the input data D and a general preference for simpler models. From a Bayesian perspective, we want to maximize the posterior $P(G|D)$, which is proportional to the product of the grammar prior $P(G)$ and a likelihood term $P(D|G)$. We can decompose the grammar model into a structure part G_S (representing grammar symbols and rules) and the parameter part θ_g(rule probabilities): $G = (G_S, \theta_g)$.

The model prior $P(G)$ then factorizes to $P(G_s)P(\theta_g|G_s)$, the product of priors over structure and parameters. To define the prior over the grammar structure we follow a Minimum Description Length (MDL) principle. The grammar's description length $DL(G_s)$ is calculated by a simple encoding of productions, where every occurrence of a non-terminal in a production contributes with $log\,|N|$ bits, $|N|$ being the total number of non-terminals in the grammar. Then, the structure prior is defined as $P(G_s) = e^{-DL(G_s)}$. We use symmetrical Dirichlet parameter priors, as all productions with the same LHS form a multinomial distribution.

Stolcke [170] has shown that in order to calculate the posterior over the model structure $P(G_s|D) \propto P(G_s)P(D|G_s)$, one needs to integrate over the parameter prior:

$$P(D|G_s) = \int_{\theta_g} P(\theta_g|G_s)P(D|G_s, \theta_g)d\theta_g \tag{7.1}$$

Fortunately, we can approximate this integral with the ML estimate of $P(D|G_s)$ by using the Viterbi assumption. This basically means that we assume that every input sample is generated by a single derivation tree of the grammar. The likelihood of a single input example is then the product of all rule probabilities used in the Viterbi derivation. Since Viterbi derivations and rule probabilities θ_g depend on each other, we use the Expectation-Maximization procedure to find the optimal values for θ_g. In the E-step, starting from an estimate for θ_g, the expected usage counts $\hat{c}(X \to \lambda)$ for each rule are calculated. This is done by finding Viterbi derivations for all input data and counting the number of times every rule was used. In the M-step, the rule probabilities $\hat{\theta}_g$ are re-estimated using the formula:

$$\hat{P}(X \to \lambda) = \frac{\hat{c}(X \to \lambda)}{\sum_\mu \hat{c}(X \to \mu)} \tag{7.2}$$

where μ iterates over all possible LHS choices for X. The process is iterated until convergence.

7.4.4 2D Earley Parsing

In order to find the Viterbi derivations of each input lattice in the E-step, we use a modified version of the Earley-Stolcke parser [170], which we extended from parsing strings to parsing 2D lattices. To the best of our knowledge, we are the first to create an Earley parser for two dimensional SCFGs. We provide its implementation details in the Appendix A.

Using Earley's parser instead of more common CKY parsing [211] has a number of advantages. Its worst-case complexity is cubic in the size of the input, but it can perform substantially better for many well-known grammar classes. Another appealing property is that it places no restrictions on the form of the grammar. This sets us apart from previous work which either requires the grammar to be in Chomsky Normal Form [180], or that the rules have to satisfy optimal substructure property [144].

7.4.5 Search in Model Space

In order to define a flexible search procedure, we modify the posterior calculation with a global prior weight w, which gives us control over the balance between the likelihood and the prior. Utilizing the Boltzmann's transformation, we transform the posterior maximization into an energy minimization:

$$E(G|D) = -w \log P(G) - \log P(D|G) \tag{7.3}$$

By setting w to a low value, we decrease the influence of the prior, thereby making the search procedure stop earlier. For larger values of w, we increase the tendency to generalize beyond the data. The influence of global prior weight w on induced grammar size is shown in Table 7.1.

Starting from the initial grammar, we follow a greedy best-first approach: in each iteration, every pair of non-terminals is considered for merging, and all of the candidate grammars are evaluated. The candidate with the minimum energy is accepted if it has lower energy than the current grammar. The rule probabilities are learned in each step using the EM procedure presented in 7.4.3.

The described method produced satisfactory results in our experiments. Of course, one may imagine more intricate ways of searching through the grammar space, e.g. by using a beam search or a random walk algorithm. We leave this for future work.

	Initial grammar G_0	Induced, $w = 0.3$	Induced, $w = 1.0$
$\|N\|$	126.8 ± 6.61	26.6 ± 0.89	14 ± 0.0
$\|R_h\|$	121.8 ± 6.61	65 ± 6.70	27.8 ± 2.68
$\|R_v\|$	33 ± 0.0	15.6 ± 2.60	11 ± 1.41

Table 7.1: Size comparison: initial grammar created by grammar incorporation, and two inferred grammars with prior weights of $w = 0.3$ and $w = 1.0$.

7.4.6 Final Model

The grammar resulting from the search procedure is still limited to the lattice space. To cast the grammar back in the image space, we perform two post-processing steps.

First, we collapse sequences of the same non-terminal symbol in a production to a single symbol with correspondingly modified attributes, for example:

$$X \to \lambda Y Y \mu \qquad \underset{\dashrightarrow}{Collapse} \qquad X \to \lambda Y \mu$$

$$A = \{\{s_1, y_1, y_2, s_2\}\} \qquad\qquad A = \{\{s_1, y_1 + y_2, s_2\}\}$$

Second, for every production $p = (X \to \lambda_1 \dots \lambda_k)$, we fit a $(k-1)$-variate Gaussian distribution $\phi(A) = \mathcal{N}(\bar{\mu}, \hat{\Sigma})$ to the set of its attributes $A(p) = \{\alpha_1 \dots \alpha_n\}$:

$$\bar{\mu} = \frac{1}{n} \sum_{j=1}^{n} \alpha_j \tag{7.4}$$

$$\hat{\Sigma} = \frac{1}{n-1} \sum_{j=1}^{n} (\alpha_j - \bar{\mu})(\alpha_j - \bar{\mu})^T \tag{7.5}$$

This enables us to sample productions with continuous attributes, by sampling directly from the estimated size distribution. Note that every production with the RHS size of k has $k-1$ degrees of freedom. If $k = 1$, we are dealing with a lexical production, for which no distribution is estimated since they have the relative size of 1 by definition.

7.5 Parsing in Image Space

The grammar induced in the previous section is now amenable for image-scale parsing. However, two main problems arise when trying to design an efficient

optimization method. First, we cannot use exact methods such as dynamic programming as we allow our attributes to take on continuous values. Second, due to the stochastic nature of the grammar, the number of attributes can change. In order to tackle the first issue, we use a Markov Chain Monte Carlo approach, which reduces the optimization to sampling. However, as the MCMC operates over a fixed-dimensional space, we must consider its extension in the form of Reversible jump MCMC (rjMCMC). Talton *et al.* [175] presented a rjMCMC-based method to parse parametric, stochastic, context-free grammars, given a high-level specification of the desired model. However, their method requires that only terminal symbols of the grammar contain descriptive continuous parameters. In contrast, we present a modified version of [175] that lifts this constraint. We also use a different likelihood computation, utilizing a pixel-based classifier to calculate the terminal merit.

7.5.1 Grammar Parsing via rjMCMC

For a given test image, our task is to find the derivation from the grammar that has the best fit to the image. Similarly to Sec. 7.4.3, we define a posterior of the derivation δ given the image:

$$P(\delta|I) \propto P(I|\delta) \underbrace{\prod_{s\in\delta} P(r_s) \prod_{s\in\delta} \phi(A(r_s))}_{P(\delta)} \tag{7.6}$$

where ϕ is defined in Sec. 7.4.6. Note that we have factorized the prior into a rule and attribute term over all non-terminal nodes s of the derivation tree. We can ignore the normalizing constant for the purposes of maximization and define the energy through Boltzmann's transformation:

$$E(\delta|I) = -log\, P(I|\delta) - \sum_{s\in\delta} log\, P(r_s) - \sum_{s\in\delta} log\, \phi(A(r_s))$$

$$E_\delta = E_\delta^{image} + E_\delta^{rule} + E_\delta^{attribute} \tag{7.7}$$

The energy that we want to minimize is composed of three terms. The *rule term* is calculated by summing up the negative log probabilities of all rules r_s selected in the derivation. The *attribute term* measures the discrepancy between the proposed attributes and the expected values of attribute distributions estimated in Sec. 7.4.6. To calculate the *image term*, we use the Random Forest pixel classifier [182], which outputs the label probability distribution P_{RF} for each pixel in the image.

$$E^{image} = \sum_{t\in\delta} \sum_{x_i\in t} -log\, P_{RF}(l_t|x_i) \tag{7.8}$$

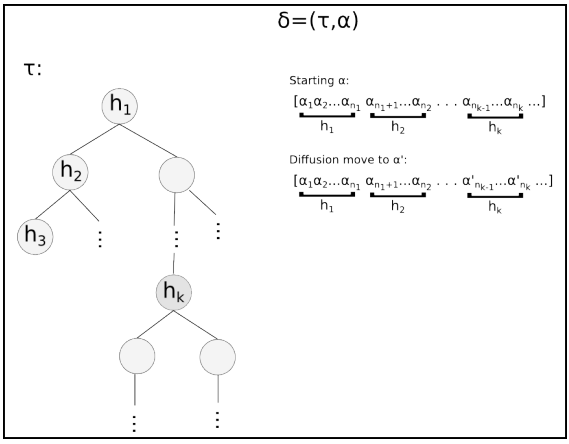


Figure 7.3: Diffusion move.

The sum is defined over all terminals t in the derivation tree. Integral images are used to rapidly calculate the inner summation of pixel energies over the rectangular region of each terminal symbol. By making this choice of image support, we can make a direct comparison to the approach of Teboul *et al.* [180].

Search. We utilize the standard rjMCMC formulation with Metropolis-Hastings (MH) update [69]. The chain is initialized with a random derivation $\delta = (\tau, \alpha)$ from the grammar. We define α as a concatenation of all selected attribute elements (i.e. relative RHS sizes) in a pre-order traversal of tree τ. To ease the discussion, we will refer to α as the *parameter vector*.

In every MH iteration, the chain is evolved by performing either a dimension-preserving "diffusion" move, or a dimension-altering "jump" move [175].

In the **diffusion** move, a random node h_k is selected in the tree. The parameters corresponding to that node in the tree are then resampled from independent Gaussian proposal distributions. The parameters at other nodes remain unchanged. The acceptance probability for a move from state x to state y is given by [26]:

$$\rho_{x \to y} = min\{1, \frac{p(y)q(x|y)}{p(x)q(y|x)}\} \tag{7.9}$$

In our case, the acceptance probability for a diffusion move from a derivation δ to δ' is:

$$\rho_{\delta \to \delta'} = min\{1, \frac{p(\delta'|I)q(\delta|\delta')}{p(\delta|I)q(\delta'|\delta)}\} \tag{7.10}$$

Since the derivation tree remains unchanged, we can write the proposal distribution q as:

$$\begin{aligned} q(\delta'|\delta) &= q(\alpha'|\alpha) \\ &= q(\alpha'_{n_{k-1}}|\alpha_{n_{k-1}})q(\alpha'_{n_{k-1}+1}|\alpha_{n_{k-1}+1})\dots q(\alpha'_{n_k}|\alpha_{n_k}) \\ &= \mathcal{N}(\alpha'_{n_{k-1}};\alpha_{n_{k-1}},\sigma^2)\cdot\mathcal{N}(\alpha'_{n_{k-1}+1};\alpha_{n_{k-1}+1},\sigma^2)\cdot\ldots\cdot\mathcal{N}(\alpha'_{n_k};\alpha_{n_k},\sigma^2) \\ &= \mathcal{N}(\alpha_{n_{k-1}};\alpha'_{n_{k-1}},\sigma^2)\cdot\mathcal{N}(\alpha_{n_{k-1}+1};\alpha'_{n_{k-1}+1},\sigma^2)\cdot\ldots\cdot\mathcal{N}(\alpha_{n_k};\alpha'_{n_k},\sigma^2) \\ &= q(\alpha_{n_{k-1}}|\alpha'_{n_{k-1}})q(\alpha_{n_{k-1}+1}|\alpha'_{n_{k-1}+1})\dots q(\alpha_{n_k}|\alpha'_{n_k}) \\ &= q(\alpha|\alpha') = q(\delta|\delta') \end{aligned} \tag{7.11}$$

The proposal distribution is symmetric, so the acceptance rate simplifies to:

$$\rho_{\delta\to\delta'} = min\{1, \frac{P(\delta'|I))}{P(\delta|I)}\} = min\{1, e^{-(E_{\delta'}-E_\delta)}\} \tag{7.12}$$

In the **jump** move, a random node h is selected from the tree, and a new rule is sampled from all rules applicable to the current LHS. If the size of the RHS changes, the entire subtree of h has to be rederived. This changes the topology of the tree, as well as the dimension of the parameter vector.

Following the discussion by Green [70], we denote the current state of the chain with $x = (k, \alpha)$, where k denotes the current space of dimension n_k and α the current set of parameters. From the current state, a move m is attempted by sampling a random vector u of r_m random numbers from a known density g_m. The new state $x' = (k', \alpha')$ can be constructed with a deterministic function h such that $(x', u') = h_m(x, u)$. Here, u' is a r'-dimensional vector of random numbers sampled from a known density g', needed for the reverse move from x' to x : $(x, u) = h'(x', u')$. The transformation between (x, u) to (x', u') must be a diffeomorphism, i.e. both h and its inverse h' are required to be bijections and differentiable. This leads to a 'dimension-matching' constraint: $n_k + r_m = n_{k'} + r'_m$.

The acceptance rate for the move m is given by:

$$\rho_{x\to x'} = min\{1, \frac{\pi(x')}{\pi(x)}\frac{j_m(x')}{j_m(x)}\frac{g'_m(u')}{g_m(u)}\left|\frac{\partial(\alpha', u')}{\partial(\alpha, u)}\right|\} \tag{7.13}$$

where π represents the target density, $j_m(x)$ is the probability that the move m is attempted at state x, and the last factor represents the Jacobian of the mapping from (α, u) to (α', u').

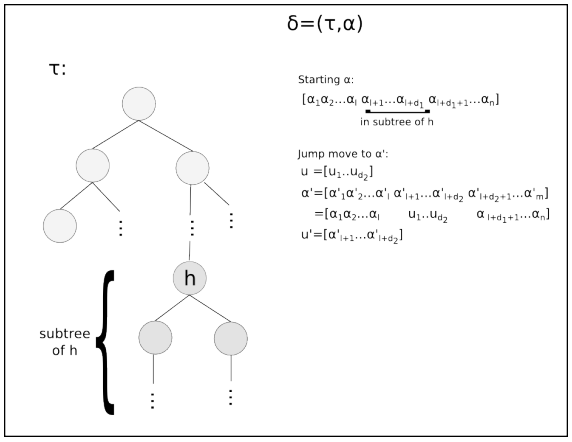


Figure 7.4: Jump move.

We shall now fully define the jump move. Since in our case the state of the chain is defined as $\delta = (\tau, \alpha)$, the dimension of the initial space n_k is the dimension of the parameter vector α. Let $l+1$ be the index of the first parameter of node h in the concatenated vector α, d_1 the number of parameters in the subtree τ_h underneath the node h, and d_2 the number of parameters in the subtree τ'_h after resampling the rule at h. We can define u and u' as vectors of d_2 and d_1 uniformly sampled numbers in the interval $[0, 1]$, respectively. In terms of the previous discussion, $r_m = d_2$, $r'_m = d_1$, and $g_m(u) = \mathcal{U}_{[0,1]}(\mathrm{u})$.

We can now fully write the mapping h_m as follows:

$$\alpha'_i = \begin{cases} \alpha_i & , i \in [1, l] \\ u_{i-l} & , i \in [l+1, l+d_2] \\ \alpha_{i-d_2+d_1} & , i \in [l+d_2+1, m] \end{cases} \tag{7.14}$$

$$u'_i = \alpha_{i+l} \quad , i \in [1, d_2] \tag{7.15}$$

The reverse mapping is obtained from Eq. 7.15 by swapping (α, u, m, d_2) with (α', u', n, d_1). It is clear that by using this mapping, the Jacobian in Eq. 7.13 reduces to unity. Since the proposal distributions g_m are uniform on the interval $[0, 1]$, the third term in Eq. 7.13 also vanishes. We define the probability of the move j_m from $\delta = (\tau, \alpha)$ to $\delta' = (\tau', \alpha')$ as:

$$j(\delta'|\delta) = q_\tau(h) \prod_{s \in \tau'_h} P(r_s) \tag{7.16}$$

where $q_\tau(h)$ is the probability of selecting a nonterminal h in the tree τ. The second term evaluates the probabilities of rules selected in the derivation of the newly derived subtree τ'_h.

Now we can write the final acceptance probability as:

$$\begin{aligned}
\rho_{\delta\to\delta'} &= min\{1, \frac{P(\delta'|I)}{P(\delta|I)} \frac{j(\delta|\delta')}{j(\delta'|\delta)}\} \\
&= min\{1, \frac{P(\delta'|I)}{P(\delta|I)} \frac{q_{\tau'}(h)\prod_{s\in\tau_h} P(r_s)}{q_\tau(h)\prod_{s\in\tau'_h} P(r_s)}\} \\
&= min\{1, \frac{P(I|\delta')\prod_{s\in\tau'} P(r_s)\prod_{s\in\tau'}\phi(A(r_s))}{P(I|\delta)\prod_{s\in\tau} P(r_s)\prod_{s\in\tau}\phi(A(r_s))} \frac{q_{\tau'}(h)\prod_{s\in\tau_h} P(r_s)}{q_\tau(h)\prod_{s\in\tau'_h} P(r_s)}\} \\
&= min\{1, \frac{q_{\tau'}(h)}{q_\tau(h)} \frac{P(I|\delta')\prod_{s\in\tau'}\phi(A(r_s))}{P(I|\delta)\prod_{s\in\tau}\phi(A(r_s))}\} \\
&= min\{1, \frac{q_{\tau'}(h)}{q_\tau(h)} \frac{e^{-E^{image}_{\delta'}}\cdot e^{-E^{attrs}_{\delta'}}}{e^{-E^{image}_{\delta'}}\cdot e^{-E^{attrs}_{\delta'}}}\} \\
&= min\{1, \frac{q_{\tau'}(h)}{q_\tau(h)} e^{-[(E^{img}_{\delta'}+E^{attr}_{\delta'})-(E^{img}_{\delta}+E^{attr}_{\delta})]}\} \qquad (7.17)
\end{aligned}$$

where we have utilized Eq. 7.6 and Eq. 7.7 to factorize the posterior and to write the final expression in terms of energies instead of probabilities.

Additional remarks. The chain is guaranteed to converge to the true posterior as the number of iterations goes to infinity. In practice, the random walk is stopped after a certain number of iterations. Similar to Talton *et al.* [175], we use parallel tempering to improve the speed of convergence. Eight chains are run in parallel, with temperature quotient between chains set to 1.3. For jump moves, we employ the technique of *delayed rejection*: a diffusion move is attempted immediately after a jump move, and two moves are accepted or rejected in unison.

7.6 Results

In all grammar learning experiments, the training set was limited to 30 images to keep the induction time within reasonable bounds. In image parsing experiments, w is set to 0.3, and rjMCMC search is run for 100k iterations. The process is repeated 5 times, and the minimum energy chain state is selected as the result.

Class	RF[182]	RL[180]	Ours	[113]
Window	29	62	66	75
Wall	58	82	80	88
Balcony	35	58	49	70
Door	79	47	50	67
Roof	51	66	71	74
Sky	73	95	91	97
Shop	20	88	81	93
Overall	48.55	74.71	74.82	84.17

Table 7.2: Per-class and overall pixel parsing accuracy (in percent) on the ECP dataset: RF - Random Forest. RL - Manually designed grammar.

7.6.1 Parsing Existing Facades

To show that our grammar learning is usable on real-world examples, we use the well-established Ecole Centrale Paris (ECP) facade parsing dataset [178], which contains 104 images of Haussmannian-style buildings. We use the same 5-fold cross-validation experimental setup as in Chapter 5.

In Table 7.2 we compare the accuracies achieved by four different semantic facade segmentation methods. Each method was evaluated on the ground truth annotations from [113]. We evaluate the accuracy in terms of class-wise and total pixel averages. As a baseline, we use the MAP estimation of the Random Forest classifier [182]. Our approach clearly outperforms the baseline in the total pixel accuracy and all but one class. Since the RF classifier output is used in both our method and the RL-based approach of Teboul *et al.* [179], our methods are directly comparable. The results that we obtain show that learned grammars can be just as effective in facade parsing as their manually written counterparts, even outperforming them in some cases. Some of the parsed images can be seen in Fig. 7.5 and Fig. 7.7.

To put the results in context, we also show the performance of the state of the art (SOA) method[2] in facade parsing [113]. However, as the SOA method uses segment classification and object detectors, it is not strictly comparable, since here we use only pixel classification cues.

7.6.2 Generating Novel Designs

The advantage of having a grammar for a certain style of buildings is that we can easily sample new designs from it. In this scenario, we generate a

[2]At the time of writing of this chapter.

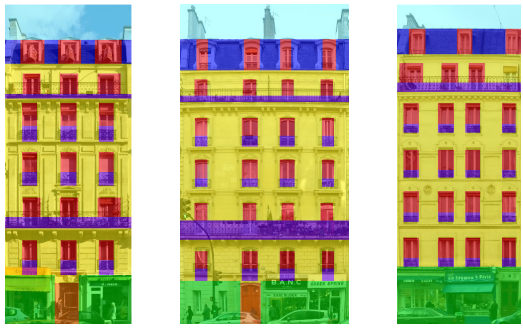
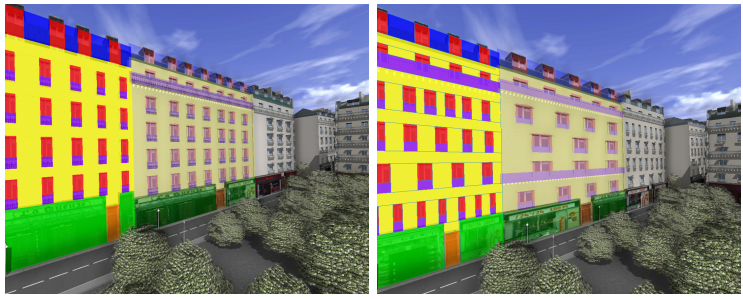


Figure 7.5: Example images from the ECP dataset parsed with our induced grammar. Note that the output is not restricted to a grid as in [180].



(a) (b)

Figure 7.6: Generating a scene with different grammars. (a) Samples from the Bayes-optimal grammar. (b) The grammar is over-generalizing due to high prior weight.

random derivation from the grammar by starting from the grammar axiom as the first node of the tree. At each node, we sample a rule based on its probability in the grammar. The relative sizes of the RHS are sampled from the estimated Gaussian distribution ϕ. Finally, the terminal symbols are replaced with instances of architectural elements from a 3D shape and texture library. We rendered a whole street of buildings sampled from our induced grammar in CityEngine [42]. The results are shown in Fig. 7.6, where we also demonstrate the effect of the prior weight parameter w on the generalization capabilities of the grammar. In Fig. 7.6b, we had intentionally set the prior weight too high, hence all compatible non-terminal symbols were merged, leading to an excessively general grammar. With the proper choice of w, we can find a good trade-off between the data fit and generalization, as shown in Fig. 7.6a.

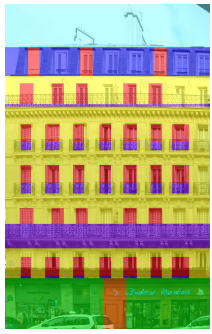
7.7 Conclusion

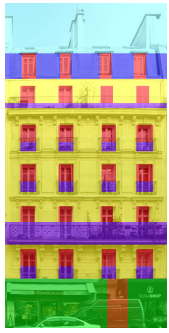
In this chapter we introduced a principled way of learning procedural split grammars from labeled data. The validity of our approach is demonstrated on two applications in urban modeling. Our induced procedural grammar not only generates new buildings of the same style, but also achieves competitive results in facade parsing, outperforming similar approaches which require a manually designed set of grammar rules.

In the future, other strategies for the design of grammar merging operators could be explored, undoubtedly requiring more elaborate search strategies. Furthermore, more complex shape grammars could be inferred by extending the Earley parser, which is currently limited to grid-like designs.

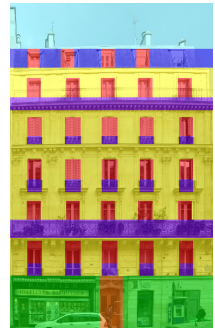
The main drawback of the proposed method is that it requires a set of manually annotated images as input. We argue that labeling a few images of a certain style and then learning the grammar is a simpler and faster approach than designing the grammar by hand. Ideally, we want to eliminate that form of supervision, and introduce robustness to noise. The next chapter therefore deals with structure learning from noisy data, allowing us to replace human annotation with automatic labeling approaches such as the one introduced in Chapter 5.

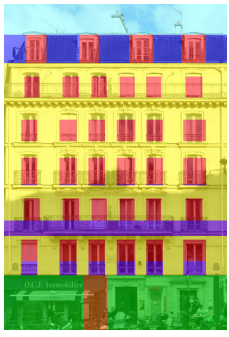
Since the time of writing of this chapter, several other authors have presented alternative ways to learn split grammars. In a concurrent work to ours, Weissenberg *et al.* [198] propose an approach which generates binary split trees from each facade and then recursively merges production rules into complex ones. However, they do not perform experiments on using the generated grammars for image parsing. Recently, Gadde *et al.* [57] has presented a method which starts from a 'simple-to-write' manually designed grammar. This grammar is used to parse the input images, thus generating initial parse trees. An unsupervised clustering problem then infers a specific grammar which can parse images of buildings from different architectural styles, including the newly introduced Paris Art Deco dataset [56].


monge_1


monge_3

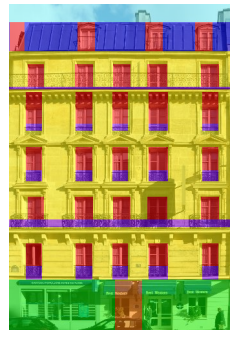

monge_6

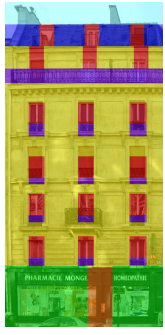

monge_9


monge_29bis

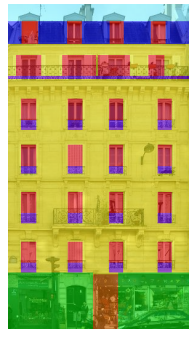

monge_61


monge_65


monge_71


monge_74


monge_79bis


monge_86

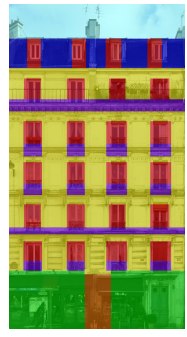

monge_101

Figure 7.7: Additional parsing results with the learned grammar from one fold of the ECP dataset.

Chapter 8

Hierarchical Co-Segmentation of Building Facades

In this chapter[1], we introduce a new system for automatic discovery of high-level structural representations of building facades. Our main assumption is that each facade can be represented as a hierarchy of rectilinear subdivisions, which is a common premise, especially when using split grammars, see Sec. 2.1.3. Under this assumption, the goal is to find the optimal direction of splitting, along with the number and positions of the split lines at each level of the tree. Unlike previous approaches, where each facade is analysed in isolation, we propose a joint analysis of a set of facade images. To achieve this, a co-segmentation approach is used to produce consistent decompositions across all facade images. Afterwards, a clustering step identifies semantically similar segments. Each cluster of similar segments is then used as the input for the joint segmentation in the next level of the hierarchy. We show that this approach produces consistent hierarchical segmentations on two different facade datasets. Furthermore, we argue that the discovered hierarchies capture essential structural information, which is demonstrated on the tasks of facade retrieval and virtual facade synthesis.

[1]This chapter is based on the joint work with Luc Van Gool, published in 3DV 2014 [115]. Anđelo Martinović is the first author.

8.1 Introduction

As mentioned in the previous chapter, there have been several attempts to *learn* facade structure from data. One idea is to assume regularity in the input. For example, Muller *et al.* [126] estimate a regular grid of facade elements and convert it into a shape grammar. Shen *et al.* [154] extend this approach by modeling multiple interlaced grids. Other facade structure learning methods typically require user interaction [128, 7, 110]. Newer approaches require pre-defined abstractions of the input facades in form of labeled boxes [213] or pixelwise annotations. Two examples of the latter are the approach we presented in the previous chapter, or the concurrent work of Weissenberg *et al.* [198].

In this chapter, we propose to merge the gap between the supervised facade parsing techniques, which produce imperfect labelings, and the structured learning approaches that require clean data to work. Due to the existence of noise in the data, we argue that the structure cannot be reliably estimated from a single facade image, and thus propose a joint optimization of a set of facade images. We create consistent hierarchies by performing a joint segmentation of facades and their parts recursively, see Fig. 8.1 for an illustration. The joint segmentation at each level is performed using a modified linear programming technique originally introduced in [81] for 3D shape segmentation.

The two existing approaches most similar to the one presented in this chapter are the works of Shen *et al.* [154] and Van Kaick *et al.* [194]. In the first approach, a hierarchy is created for each facade independently, resulting in less stable hierarchies with no correspondence across images. In the second approach, a co-analysis is performed on a set of 3D shapes, however with the key difference that the hierarchies are first created independently for each shape and subsequently merged. Furthermore, they are limited to an analysis of binary trees, while our approach allows us to create n-ary trees from the outset.

8.2 Overview

We start with a set of N facade images $\mathbf{F}$ and their noisy semantic segmentations (labelings) $\mathbf{L}$. We obtain the latter by running the first two layers of our three-layered approach (Chapter 5), which provide labeling results with high accuracy, without introducing any explicit architectural knowledge. Other approaches for semantic segmentation could also be used [35, 30]. Our main premise is that the facades follow the Manhattan-world assumption, and can be decomposed by recursive splitting in the vertical and horizontal direction. This is a common assumption used by a large number of previous works in

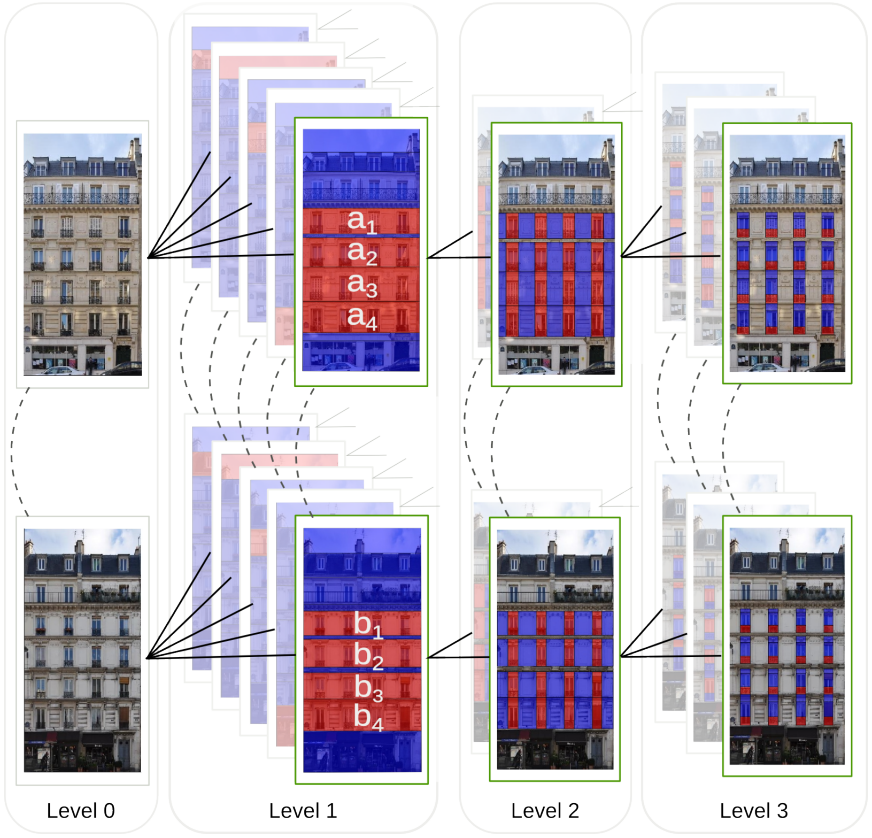


Figure 8.1: An example hierarchical joint segmentation of two facade images. Solid lines represent the hierarchical decomposition. Dashed lines indicate which segments are used in a joint segmentation. For example, one joint segmentation is performed for the initial facades, and one for segments $\{a_i \cup b_i\}$.

urban modeling [202, 154, 35, 181] which does not preclude the existence of non-rectangular elements (e.g. round windows) since they can still be represented with a bounding box. We define a *scope* z as an axis-aligned bounding box which contains a non-empty area of an image and its corresponding labeling. The initial set of images is thus converted to a set of N scopes $\mathbf{Z} = \{z_i\}$, each scope completely covering one facade image.

At every step of the hierarchy, we want to find the optimal segmentations of all scopes, such that the created segments are consistent across scopes. Due to the Manhattan assumption, we can restrict our search by considering only segments generated by splitting the scope in one of the main splitting directions. A valid k-way segmentation of a scope thus consists of k adjacent segments separated by $k-1$ splitting lines. A brute-force approach to finding the consistent segmentations would be to consider every possible combination of split lines in each scope, and selecting the combination which maximizes some predefined consistency score. Since this would be too computationally expensive, we reduce the dimensionality of the problem by limiting the number of allowed split line positions. This is done by first generating an oversegmentation of each scope into a large number of smaller segments, or *slices*, and constraining each segment to be a superset of contiguous slices (see Fig. 8.2). This idea is similar to using superpixels [142] in general image segmentation, or patches in shape segmentation [81].

The optimal subsets of segments for each scope are then selected by a modified co-segmentation approach of Huang *et al.* [81], detailed in Sec. 8.4. Then, a hierarchical decomposition is created with a recursive approach detailed in Sec. 8.5. Similar segments across scopes are discovered in a graph clustering step. All segments in one cluster are used as the input to the joint segmentation stage in the next level of the hierarchy. The process continues until the produced clusters contain uniform elements (e.g. wall regions) or elements too small for subdivision. In Sec. 8.6 we show that the resulting hierarchies can be used for structural facade retrieval and sampling of virtual facades.

The contributions of this chapter are as follows:

1. A novel approach for creating consistent hierarchical decompositions of building facades. To the best of our knowledge, we are the first to use a co-segmentation approach in this context;

2. A graph clustering approach for automatic discovery of semantically similar elements across images;

3. A new tree distance measure for comparing n-ary trees based on sequence matching.

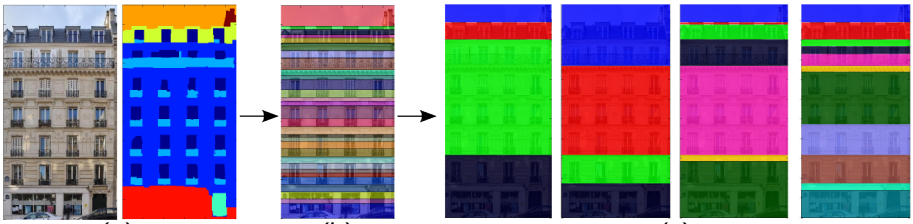


Figure 8.2: A single image-labeling pair (a) is oversegmented into a large number of slices (b). Randomized segmentations (c) with a varying number of segments create the initial pool of segments.

8.3 Initial Segmentation

In order to generate an oversegmentation of a scope, we define a support function for placing a split line at each position in the scope:

$$\Upsilon(z) = \Upsilon_{IG}(\mathbf{F}_z) \cdot \Upsilon_{IC}(\mathbf{F}_z) \cdot \Upsilon_{LB}(\mathbf{L}_z) \cdot \Upsilon_{LC}(\mathbf{L}_z) \tag{8.1}$$

This function aggregates the data support from both the original image and the noisy labeling through the following four factors, normalized to the interval $[0, 1]$:

- **Image gradient support** Υ_{IG} [154, 35] promotes placing of horizontal (vertical) split lines where horizontal (vertical) edges or gradients are prominent, and vertical (horizontal) edges are rare.
- **Image content support** Υ_{IC} [35] proposes split line positions based on the inverse of the normalized cut between the two created parts of the image.
- **Label border support** Υ_{LB} uses the semantic information from the labeling to penalize lines which split facade elements such as windows, doors and balconies.
- **Label content support** Υ_{LC}: same as Υ_{IC}, but defined over the labeled image.

The next step is to create an oversegmentation of a scope into a predefined number of slices. We are looking for at most K split lines (30 in our experiments), corresponding to peaks in the data support function, which split the scope into $K + 1$ slices.

A peak in a vector is defined as a position where the vector has a higher value than its neighbors, and is preceded by a value lower than a threshold τ. By setting τ to a very low value, we initially detect a large number of peaks, most of them affected by the noise in the support function. However, we can smooth the support function by convolving it with a Gaussian window, thus reducing the total amount of detected peaks. The peak detection problem is now posed as the search for the best Gaussian window which produces a number of peaks as close as possible to K.

We solve this problem using binary search, setting the initial lower and upper bound on the Gaussian window size to $\gamma_l = 1$ and $\gamma_u = |\Upsilon(z)|$ respectively. In each step, we convolve the support function with the Gaussian window of size $\gamma_m = (\gamma_u + \gamma_l)/2$. If we detect more peaks than $K - 1$, the search is continued by setting $\gamma_l = \gamma_m$. If the number of peaks is smaller, we set $\gamma_u = \gamma_m$. The algorithm finishes when $\gamma_l = \gamma_u$ or the number of generated peaks is equal to K. The result of this step is a set of $K + 1$ slices $\mathbf{C}_z$, and K splitting lines $\mathbf{l}_z$, for each scope z. The support function evaluated at the splitting line positions $\Upsilon(z, l_j)$ gives us the strength of each split line, which we use to group the slices into segments.

8.3.1 Segment Proposals

Similar to Huang *et al.* [81], from a set of slices $\mathbf{C}_z$, we generate many proposal segmentations by varying the number of target segments k from 1 to 20, and running 250 rounds of randomized segmentations for each k. In each round, we perform k-medoid clustering of slices, following the EM pattern. We initialize the algorithm by uniformly sampling k slices as cluster centers. In the M-step, every slice $c_i \in \mathbf{C_z}$ is assigned to the closest cluster center c_m, based on the distance between two slices $\delta(c_1, c_2)$. We define this distance as the maximum of all split line strengths Υ between two slices, which penalizes the creation of segments which span strong split lines. If there is another cluster center c'_m between c and c_m, the distance $\delta(c, c_m)$ is set to infinity. This forces the segmentation to contain only contiguous clusters of slices. In the E-step, the medoids are estimated from the cluster members, by minimizing the sum of distances between elements in one cluster.

Typically, many segments generated in this fashion will appear in more than one randomized segmentation. Segments that are generated the most often are the ones most useful to us, being less sensitive to randomization and the selected number of target segments. Therefore, we weight each unique segment s by the number of times it appears over all randomized segmentations of a single scope, i.e. the frequency of appearance is used as the fitness score w_s of

the segment. This simple weighting scheme proved to be sufficient for the task, as we did not observe any improvement by using the more elaborate weighting scheme from Huang *et al.* [81]. Finally, to reduce the total amount of segments for the subsequent optimization procedure, for each scope we retain $n = 100$ segments with the highest weight as the set of proposals I_z, making sure that it contains at least one complete segmentation of the scope.

We represent each segment s with a vector $\mathbf{h}(s)$ which is a concatenation of two types of features:

- **Label features** $\mathbf{h}^l(s)$. Histogram of labels from the entire segment and from each of its $2x2$ subdivisions. The resulting feature vector captures the coarse distribution of labels.
- **Image features** $\mathbf{h}^i(s)$. Histogram of visual words. Dense SIFT features are extracted from all images, followed by K-means clustering into a codebook of 256 visual words.

Finally, to measure the dissimilarity between two segments, we introduce a distance measure based on the histogram intersection between the feature vectors:

$$d(s, s') = 1 - \frac{\sum_i min(\mathbf{h}_i(s), \mathbf{h}_i(s'))}{\sum_i \mathbf{h}_i(s')} \tag{8.2}$$

8.4 Co-Segmentation

The purpose of the co-segmentation step is to find the best subset of segments for each scope, such that they are salient in each scope and consistent across scopes. We follow the same basic algorithm which was introduced for joint segmentation of 3D shapes [81]. In this section, we summarize the basic algorithm, with emphasis on the main differences introduced in our work.

8.4.1 Pairwise Co-Segmentation

Given two scopes z_1 and z_2 and their corresponding sets of proposal segments I_1 and I_2, the pairwise co-segmentation searches for the best valid subsets of segments $S_1 \subseteq I_1$ and $S_2 \subseteq I_2$, by maximizing both the quality of individual segmentations, and the consistency between them. A subset of segments is considered valid only if the selected segments cover the entire scope without

overlapping. The consistency between two scopes is modeled through two many-to-one mappings $M_{ij} \subset S_i \times S_j$, from segments in S_1 to segments in S_2, and vice versa. The many-to-one mappings allow us to match scopes with different amount of corresponding parts. Thus, each segment in one scope will be mapped to at most one segment in the other scope. The maximization can be written as

$$\max_{S_1,S_2,M_{12},M_{21}} \sum_{s \in S_1 \cup S_2} r_s w_s + \lambda \sum_{(s,s') \in \{M_{12},M_{21}\}} r_s w_{(s,s')} \tag{8.3}$$

where the parameter λ (0.1 in our experiments) weighs the relative importance of the segmentation (left) and consistency scores (right). The segmentation score is a normalized sum of segment weights w_s, defined in Sec. 8.3.1. The normalization factor is the relative size of the segment s in the scope z: $r_s = \text{area}(s)/\text{area}(z)$.

The consistency term is a normalized sum of similarity weights between all segment pairs (s, s') induced by each of the two mappings. The similarity is determined based on the distance measure d between two segments (Sec. 8.3.1):

$$w_{(s,s')} = exp\left(-\frac{d^2(s,s')}{2\sigma^2}\right) \tag{8.4}$$

In our experiments, σ is set to half the maximum distance between all pairs of most similar segments.

Integer Programming Formulation

The maximization problem from Eq. 8.3 can be reformulated as an integer program [81]. For every segment $s \in I_i$, an indicator variable x_s is introduced, and defined to be $x_s = 1$ when the segment is selected, and 0 otherwise. Additionally, for every pair of segments $(s, s') \in I_i \times I_j$, the indicator variable $y_{(s,s')}$ is defined to be 1 when this pair is selected in the mapping M_{ij}. The objective function from Eq. 8.3 is then reformulated as follows:

$$\max \sum_{i \in \{1,2\}} \mathbf{x}_i^T \mathbf{w}_i^{seg} + \lambda \sum_{ij \in \{12,21\}} \mathbf{y}_{ij}^T \mathbf{w}_{ij}^{cor} \tag{8.5}$$

where $\mathbf{x}_i$ and $\mathbf{w}_i^{seg}$ represent all segment indicators in I_i and their normalized weights. Likewise, $\mathbf{y}_{ij}$ is a binary vector of all pair indicators in $I_i \times I_j$, and $\mathbf{w}_{ij}^{cor}$ are their normalized similarity weights. The first set of constraints in the integer program states that the selected segments must cover the entire scope z_i, without overlapping:

$$\sum_{s \in cover(c)} x_s = 1 \quad \forall c \in \mathbf{C}_{z_i} \tag{8.6}$$

where $cover(c)$ is the set of all segments that contain slice c. Secondly, each segment of I_i can map to at most one segment in I_j, which itself has to be selected:

$$\sum_{s' \in I_j} y_{(s,s')} \leq x_s \quad \forall s \in I_i \tag{8.7}$$

$$y_{(s,s')} \leq x_{s'} \quad \forall (s, s') \in I_i \times I_j \tag{8.8}$$

The integer problem (IP) is obtained by adding the integrality constraints on the x and y variables. Note that our program contains $2n + 2n^2$ integer variables, unlike [81], where adjacency constraints create $2n^4$ additional variables. In our experiments, using the adjacency term did not result in any noticeable improvement. Another difference is that we solve the IP by relaxing the integrality constraints only on the y variables, resulting in a mixed-integer linear program (MILP) with $2n$ integrality constraints. This approach gives us a tighter bound on the IP solution than we would obtain by relaxing all variables. Although the worst-case complexity of this MILP is $O(2^{2n})$, in practice the constraints (8.7) and (8.8) allow for a quick convergence of branch-and-cut methods, such as the MOSEK MILP solver in the CVX software package [68]. For $n = 100$, the optimal solution is usually reached within a few seconds on an 8- core machine. The fractional y variables are subsequently rounded to the closest integer, respecting the constraints.

Segment filtering. After the optimization in Eq. 8.5 has been performed for every pair of scopes, we introduce an additional filtering step. For each scope, we keep only the segments that are selected in at least one of the pairwise optimizations. By discarding the remaining segments, we reduce the computational burden in the subsequent stages.

8.4.2 Multiway Co-Segmentation

A joint segmentation of all scopes is performed by a generalization of Eq. 8.5 to N scopes:

$$\max \sum_{i=1}^{N} \mathbf{x}_i^T \mathbf{w}_i^{seg} + \frac{\lambda}{N-1} \sum_{\substack{i=1 \\ i \neq j}}^{N} \sum_{j=1}^{N} \mathbf{y}_{ij}^T \mathbf{w}_{ij}^{cor} \tag{8.9}$$

Note that the segment filtering step reduces the size of $\mathbf{w}$ vectors compared to Eq. 8.5. The resulting optimization is again solved with CVX, but due to its higher complexity, we constrain the maximum run-time of the solver to

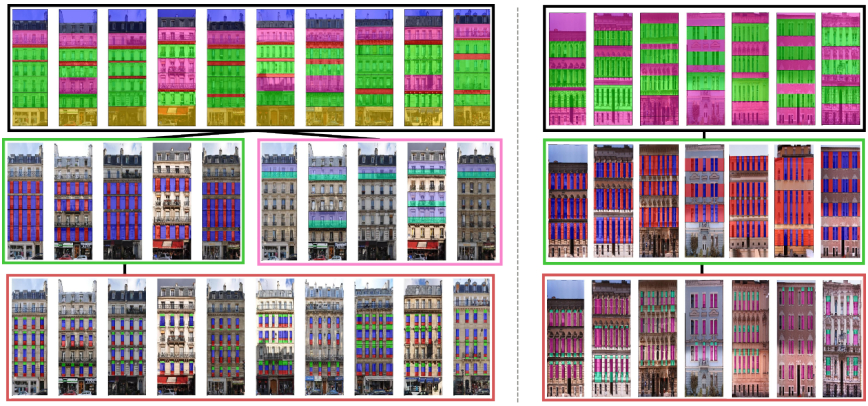


Figure 8.3: Hierarchical joint segmentation of facade images: (a) ECP dataset. (b) Gruenderzeit dataset. Each cluster of similar elements in one level of the hierarchy is represented with the same overlay color, which corresponds to the border color in the next level. Due to space restrictions, only a subset of the hierarchy is shown.

30 minutes. In our experience, this was enough to reach a solution with a sufficiently small optimality gap. An approximate block-coordinate procedure as in [81] could be employed to increase the speed of optimization, but with no optimality guarantees.

8.5 Hierarchical Co-Segmentation

The co-segmentation step results in a flat segmentation of each scope, with mappings between corresponding segments in different scopes. The next step is to create a hierarchical decomposition of each facade. The first step towards this goal is finding subsets of semantically identical elements, which can be segmented jointly in the next step of the hierarchy.

8.5.1 Segment Clustering

We represent the segments selected in all scopes by the joint segmentation into one directed, weighted assignment graph $G = (V, E)$. Each node in the set V corresponds to a selected segment s_i, i.e. a segment for which $x_i = 1$. E is a set of directed edges, where node v_i is connected to v_j if the value of the

corresponding y_{ij} variable is equal to 1. The weight of each edge e_{ij} is defined in Eq. 8.4.

Our key observation is that the groups of similar elements will form dense clusters in the graph G. By discovering these clusters, we will also find self-similarities in the input scopes, a feature not modeled by the joint segmentation itself. Thus, our goal is to determine the groups of segments which correspond to each other, within and across scopes. To this end, we run spectral clustering [197] on the graph G. We calculate the normalized graph Laplacian [155], and use its eigenvalue decomposition to find the number of clusters κ. Based on the eigengap heuristic, we sort the eigenvalues λ_i in ascending order, and pick κ as $argmax_i(\lambda_{i+1} - \lambda_i)$.

There is a possibility that after the clustering step, two neighboring segments in a scope are assigned to the same cluster, e.g. two wall parts next to each other. In these cases, we simplify the final segmentation by merging those segments into one. However, this must not be done indiscriminately, since there are cases when we expect neighboring segments of the same class (e.g. two floors). We merge two neighboring segments only if their potential merger has a small distance d to the remaining segments in the cluster.

8.5.2 Hierarchy Creation

Initially, N segmentation trees are created, each containing a single root node, corresponding to the whole facade. After performing the co-segmentation and clustering, every tree is augmented with κ children nodes, one for each cluster. In Fig. 8.3 the trees are merged and the same-cluster segments overlaid with the same color. These segments now become new sets of scopes for the next level of joint segmentation, performed recursively on each cluster. The recursion stops when either the average scope size in the direction of splitting is smaller than a predefined size, or the scopes in the set are uniform in appearance.

The direction of splitting for each node in the hierarchy is determined adaptively. We perform the joint segmentation in both directions, and select the one which gives a more consistent joint segmentation, based on the similarity between a pair of scopes z_i and z_j [81]:

$$w(z_i, z_j) = \mathbf{y}_{ij}^T \mathbf{w}_{ij}^{cor} + \mathbf{y}_{ji}^T \mathbf{w}_{ji}^{cor}, w \in [0, 2] \tag{8.10}$$

8.5.3 Segment Synchronization

As we go deeper in the hierarchy, the number of scopes to be jointly segmented increases dramatically (e.g. 20 facades result in ~80 floors and ~400 windows). We can reduce the computational burden for the joint segmentation steps further down in the hierarchy by making the following observation: within one cluster, two scopes with a common parent node are more alike than scopes originating from different parents. Therefore, instead of considering each of these scopes separately, we perform segment synchronization. First, for each set Ψ of same-cluster scopes originating from the same node, we average their data support functions:

$$\boldsymbol{\Upsilon}^{avg} = \frac{1}{|\Psi|} \sum_{s \in \Psi} \boldsymbol{\Upsilon}(s) \tag{8.11}$$

and create a *representative* scope by averaging the feature vectors of all scopes in Ψ. This scope replaces all scopes in Ψ during the joint segmentation. Afterwards, the discovered segment borders are back-projected to the original scopes.

Fig. 8.1 illustrates the process of synchronization: floors a_i are synchronized: they originate from the same facade, so they are segmented in the same way. However, their hierarchies are allowed to differ. As can be seen in the next level of the hierarchy, window tiles in floors a_i are synchronized, but different from the synchronized tiles of floors b_i. This allows us to model local differences, while still correctly capturing the global correspondence.

8.6 Results

In this section we show some qualitative results of the joint hierarchical segmentation, and evaluate the approach on the task of facade retrieval. Finally, we show how virtual facade layouts can be generated from the induced hierarchy.

8.6.1 Experimental Setup

The main evaluation of our approach is performed on the well-established ECP facades dataset [178], containing 104 images of buildings in Paris. Since all facades in this dataset follow the same Haussmannian architectural style, it is an ideal candidate for our joint segmentation approach. There are 8 semantic labels in this dataset, namely $\{window, wall, balcony, door, roof, sky, shop\}$. We also test our approach on a subset of the Graz dataset from [144], consisting

of 30 images in Gruenderzeit style, annotated with a smaller set of labels: $\{window, wall, door\}$. Since we use the output of a supervised facade parsing approach [113] as the input to our approach, we are limited to the analysis of the test set. To cover the entire ECP dataset, we repeat our experiments 5 times, in each fold using different 20 images as the test set, and average the results.

8.6.2 Hierarchical Co-Segmentation

In Fig. 8.3 we visualize the results of our approach on a subset of ECP and Gruenderzeit dataset, respectively. Due to space limitations, here we show only some nodes in the hierarchy. Full results for one fold of the ECP dataset can be seen in Fig. 8.4.

The first row of Fig. 8.3 (a) shows the consistent first-level segmentation of the Parisian facades. The coloring corresponds to the different clusters discovered in the data. Our system has automatically detected 6 clusters of similar elements, roughly corresponding to the regions of sky, roof and shop, ledges (red) and two types of floors: regular (green) and floor with running balcony (purple). The consistent segmentation reveals that running balconies usually appear in the second and fifth floor, which is one of the distinguishing properties of Haussmannian architecture. We can also see that the floors with running balconies are split differently than the regular floors in the next level of the hierarchy.

In the Gruenderzeit dataset, we can solely separate floors from the wall regions, since there are only three semantic labels in the annotations. Even in this case, the hierarchical segmentation produces reasonable results, splitting floors into window tiles, which are further subdivided into window and wall regions.

8.6.3 Facade Retrieval

In facade retrieval, a query facade is presented to the system. The query is then compared to a set of known facades based on a pre-defined distance measure. These facades are then re-ranked based on their distance to the query facade, and top K ranked facades are returned as output.

In this section, we demonstrate that our hierarchical representation of facade structures can be used for retrieval of structurally similar facades, rather than those similar in local appearance. We follow the protocol for facade comparison introduced by Weissenberg *et al.* [198] for the ECP dataset. The gold standard distance δ_{GT} between two facades is defined as the total number of architectural

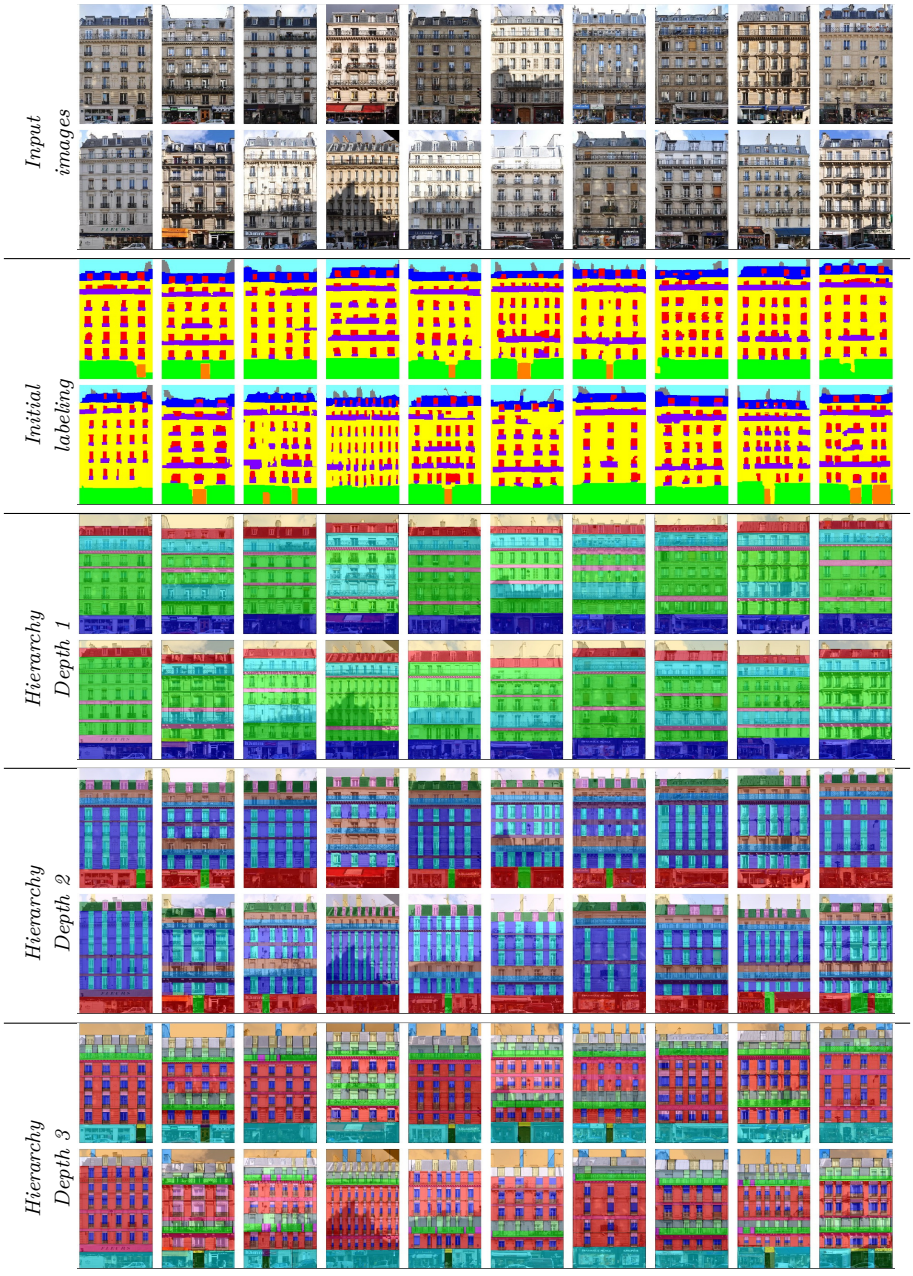


Figure 8.4: Results on the ECP dataset, fold 1. Elements from the same cluster are overlaid with the same color.

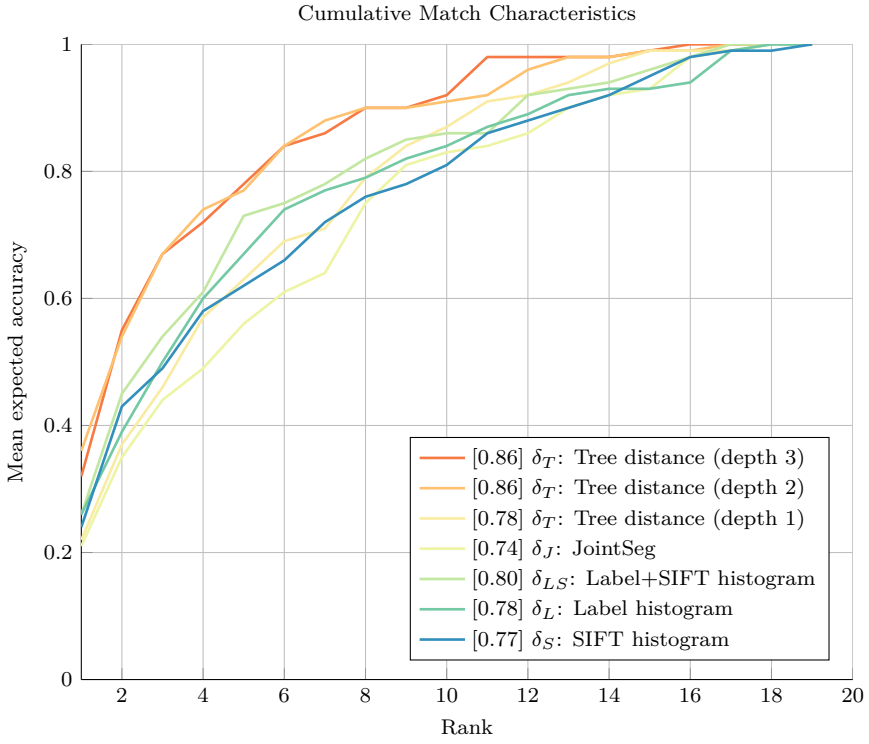


Figure 8.5: Cumulative Match Characteristics (CMC) for different retrieval methods on the ECP dataset, averaged over 5 folds. Normalized area under curve is shown in square brackets.

changes: number of floors, number of window columns, position of the running balconies and doors. For each facade in the set, all other facades are re-ranked in the ascending order of δ_{GT}, and the one with the smallest distance is kept as the ground-truth nearest neighbor. Once the ground truth ranking is established, various retrieval methods are evaluated using the Cumulative Match Characteristic (CMC). This measure counts the percentage of correctly retrieved results (gold distance nearest neighbors) in the top-K ranking. When K is equal to the dataset size, all facades are retrieved, resulting in CMC value of 1 for any method.

We test the retrieval results with several different methods, and show the results in Fig. 8.5. As the first baseline for facade comparison, we create the histograms of semantic labels in each image, and use the histogram intersection measure as the distance δ_L. Second, we use the histograms of dense SIFT features

coded into a vocabulary of 256 visual words, to obtain the distance δ_S. A combined distance δ_{LS} is calculated by concatenating the two aforementioned histograms. Additionally, we evaluate the distance $\delta_J = 2 - w(z_i, z_j)$, measuring the dissimilarity of scopes in the first-level joint segmentation step (Eq. 8.10).

In order to test our hierarchical method, we introduce a new measure for tree distance δ_T. The distance is defined recursively between two induced hierarchical segmentations h_1 and h_2 as:

$$\delta_T(h_1, h_2) = \delta_N(N_1, N_2) + \sum_{(a_1,a_2)\in A} r(a_1, a_2)\delta_T(h_{a_1}, h_{a_2}) \tag{8.12}$$

where N_1 and N_2 are the root nodes of the hierarchies, and the set A contains all pairs of children nodes that belong to the same cluster. The relative size r of the children normalizes the sum on the right-hand side to 1. We evaluate our tree distance measure δ_T with varying depth of the hierarchy. For example, the tree distance in the first level of the hierarchy is simply δ_N.

Finally, we define the node distance δ_N between the roots of (sub)hierarchies. We want the distance to be sensitive to the relative order of child nodes. Therefore, we represent the parent node as a string, where each symbol corresponds to the cluster ID of the child node. We match these two sequences using the Smith-Waterman algorithm [163], a dynamic-programming approach traditionally used in bioinformatics to align DNA and protein sequences. We simply define the cost of assigning one element of the sequence to another, and the cost of skipping an element in the sequence (gap cost). We use a constant penalty for mismatched symbols, and size-dependent gap costs, i.e. smaller facade elements are more likely to be skipped.

The retrieval results are shown in Fig. 8.5. As expected, the combination of the label histograms and SIFT features δ_{LS} outperforms either of the stand-alone methods. On the other hand, using the joint segmentation distance δ_J results in poor performance, due to two main limitations. First, this distance models only the first level of the hierarchy, and does not capture the lower-level structural differences. Second, many-to-one mappings disregard the relative frequency of elements, information which is not lost in histogram-based methods. For example, a perfect mapping can be found between a facade with 10 floors and a facade with 1 floor, resulting in a low δ_J. Our tree distance δ_T does not suffer from these issues. Even when using only one level of the hierarchy, where the only discovered elements are floors, our distance provides better discriminative power than δ_J, due to the ordering information. By using the second level of the hierarchy, we obtain even better results, due to the correct modeling of window tiles. We do not observe any significant improvement in retrieval by using the third level, where the hierarchy models only local differences. It is important to note, however, that using the deeper levels of the hierarchy does

not degrade the results, even if similarities not captured by the ground truth are found.

8.6.4 Facade Synthesis

The state-of-the-art methods for facade structure extraction [114, 198] have shown that procedural split grammars can be inferred from clean, ground-truth labelings. The grammars can subsequently be used to generate new facades by changing the parameters of the grammar. Similar to our approach, these methods split the facades in alternating vertical and horizontal directions, where each split is represented as one rule in the procedural grammar:

$$X^{\alpha} \rightarrow split(dir)\{r_1^{\alpha} : b_1^{\alpha} | r_2^{\alpha} : b_2^{\alpha} ... | r_n^{\alpha} : b_n^{\alpha}\} \tag{8.13}$$

where X^{α} represents the root node and dir the direction of splitting. With $\mathbf{b}^{\alpha} = \{b_i^{\alpha}\}$ and $\mathbf{r}^{\alpha} = \{r_i^{\alpha}\}$ we denote the set of children and the vector of their relative sizes.

However, both of the aforementioned methods create the structural decomposition of each facade separately, and then attempt to merge the inferred decompositions to obtain a joint grammar. On the other hand, our joint approach immediately creates consistent trees, albeit noisier due to the usage of imperfect input data. We create a procedural grammar from the ECP test set by transforming each of the 20 hierarchies into a set of procedural rules, which are then aggregated. In the next step, we perform the same production rule inference as in Weissenberg *et al.* [198]. This process merges all similar rules which have the same form (but different size vectors) into one. The initial facades are now re-created by selecting the appropriate size vector $\mathbf{r}^{\alpha}$. New facades in the similar style can be obtained by fitting a multivariate Gaussian distribution to the set of size vectors in each rule, and then sampling from this distribution. Fig. 8.6 shows some virtual facade layouts sampled in this fashion. We export these layouts to CityEngine [42] and create 3D buildings by automatic placement of architectural elements from a 3D library at the positions defined by the layouts.

8.7 Conclusion

This chapter introduced a system for higher-level understanding of building facades through a joint hierarchical decomposition approach. Unlike most previous facade structure learning approaches, which rely on user interaction or ground truth annotations, we show that facade structure can be induced even

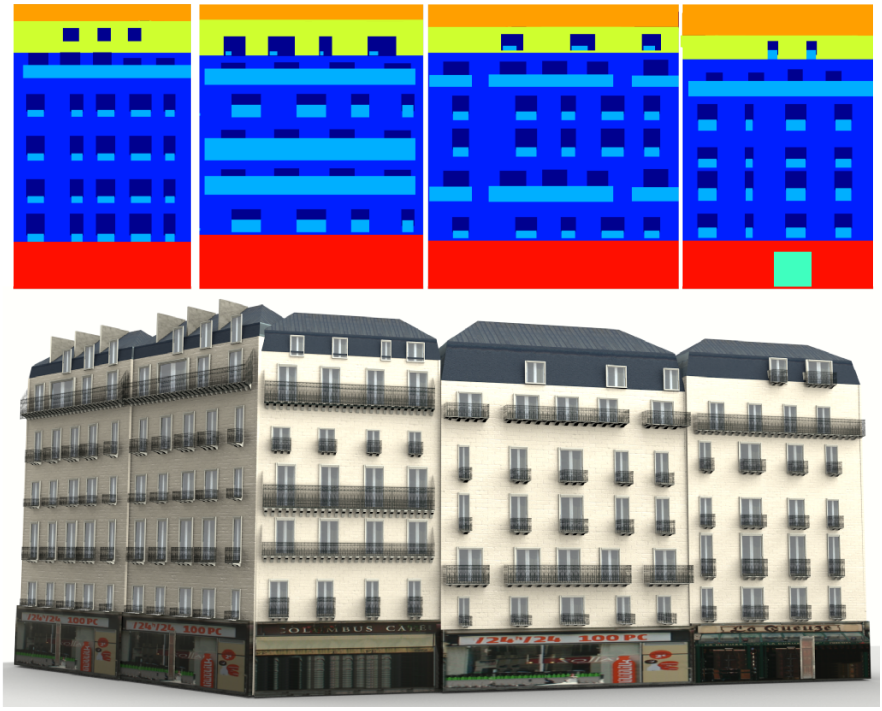


Figure 8.6: Synthesis of virtual facade layouts. The sampled layouts are represented as procedural split grammars and converted into 3D models with CityEngine.

by using noisy inputs. Our key observation is that consistent hierarchies can be created by performing a joint segmentation approach on each level of the hierarchy. The joint segmentation allows us to produce stable, consistent segmentations across images, despite the noise present in the input data. Moreover, the induced hierarchies are a meaningful semantic representation of the building facade, which we demonstrate on the task of structural facade retrieval. We also convert the hierarchies into procedural grammars and use them to sample new facade designs, which respect the layout of the original facades.

The approach presented in this chapter could be extended by adding feedback during hierarchy construction. This could help reduce the effect of error propagation to the deeper levels of the hierarchy. Although this chapter focused solely on split operations, other kinds of structural decompositions popular in hand-designed grammars may be added, such as repetition and symmetry.

Chapter 9

Conclusion

In this thesis we investigated various ways in which semantic knowledge can be used in the process of urban reconstruction. In particular, we focused on procedural models encoded as shape grammars which provide rich yet compact means to encode this knowledge. Furthermore, we investigated approaches that allow us to extract this knowledge from acquired data using techniques of inverse procedural modeling.

In Part 1 we have shown that grammars are quite useful in building reconstruction, as missing data can be filled in, detectors can be targeted to specific objects, and learn from experience by training more specific detection models. In addition, we have demonstrated that an appropriate reconstruction grammar may be selected automatically by detecting the building style. The drawback of these approaches is the dependence on scarce expert-written grammars, which hinders their large-scale applicability.

In Part 2, we pushed the limits of semantic segmentation of building facades, without a priori dependence on procedural grammars. Importantly, we have shown that generally applicable architectural principles such as symmetry or alignment can provide important cues when creating a high quality pixel-wise semantic labeling. This semantic labeling can afterwards be converted in an instance-specific procedural model. In addition, by moving the semantic segmentation process to 3D, we gain significant speed improvements (20x compared to the 2D approach), while achieving competitive perfomance. This brings us a step forward towards efficient large-scale city reconstruction with detailed facade models. The flexibility of our approach allows for higher accuracy than grammar-based approaches, although it may in some cases produce imprecise reconstructions. Thus, grammar-based approaches could still

be used when we are willing to sacrifice some of the accuracy for the guarantee that the output will be a valid facade.

In Part 3 we demonstrated that the semantic labelings of facades can be used to learn grammars from data. We have shown that split grammars can be induced from ground-truth 2D semantic labelings of same-style facades. The inferred grammars are shown to be equally effective as hand-written grammars for parsing the structure of newly observed facades. In contrast to bottom-up approaches, the induced grammars allow us to sample new facade designs in similar style, useful for example in city planning. The task of learning procedural facade grammars from data has since received increased interest [198, 57]. Finally, we have proposed an approach to induce facade structure from noisy labelings coming from bottom-up semantic segmentations. Instead of depending on manual labeling or handwritten grammars, we proposed a co-segmentation approach to directly infer consistent structures throughout the dataset. We have shown that the induced facade decompositions in form of hierarchies perform well directly in tasks such as facade retrieval, or can be converted to conventional split grammars.

9.1 Outlook and Future Work

One limitation of the grammar-based methods introduced in this thesis is the assumption that facade structure can be effectively captured by split grammars, i.e. that a facade can be represented as a sequence of splitting operations. However, in some cases, more complex operators, such as layering [213] are required to capture the architect's intent. One may also move to a different kind of grammars for facade analysis altogether, such as graph grammars [97] or 2D adjacency patterns [96].

Considering the visual quality of the produced building models, realistic texturing is an important ingredient in building reconstruction process. However, textures obtained from aerial or rectified perspective images often contain artifacts, are limited in resolution, and have large memory requirements in the final model. In order to mitigate these problems, textures of facade elements can also be generated procedurally. Some work in this vein has already appeared [106, 36]. Yet, these approaches synthesize the entire facade texture rather than separate textures for its constituent parts, and do not take into account the extra information which the semantic labels provide.

The next step in the quest for photo-realistic city rendering is the acquisition of detailed models of shape reflectance. Current state-of-the-art approaches express these material characteristics with Bidirectional Reflectance Distribution

Functions (BRDFs), which can in fact be induced from images [60]. However, applying the existing methods in outdoor scenarios with uncontrolled lighting remains a challenge.

Finally, an interesting extension of this work would be to design tools that transfer the acquired building models to widely used standards in industry, such as Industry Foundation Classes (IFC) or City Geography Markup Language (CityGML). Some authors have begun to tackle this problem by creating BIM models from raw point clouds [9] or extracting semantics from existing CAD models in IFC format [19].

Appendix A

Earley Parsing for 2D Stochastic Context Free Grammars

In Chapter 7 we have presented a novel approach of learning a specific variant of procedural grammars from data, namely 2D Attributed Stochastic Context Free Grammars, or 2D-ASCFGs. One of the essential parts of the grammar learning algorithm described therein is the Earley parser for 2D-SCFGs. The purpose of this technical report is to provide the implementation details of this parser, illustrate the approach on simple examples, and discuss the improvements over the existing methods.

A.1 Introduction

This appendix provides detailed instructions on the implementation and usage of an Earley-style parser for two-dimensional stochastic context free grammars (2D-SCFGs), used in Chapter 7. Earley's parser [41] is a well known top-down parsing algorithm for context free grammars (CFGs) that has a worst-case complexity of $O(n^3)$, where n is the size of the input. However, it was shown that the parser can perform substantially better for many well-known grammar classes. For example, it runs in quadratic time for unambiguous grammars, and in linear time for most LR(k) grammars. Another appealing property of Earley's algorithm is that it deals with any context-free rule format; it does not require

grammar conversion to Chomsky Normal Form (CNF), as some alternatives do, such as CKY [211] chart parsing. Although every context-free grammar can be transformed into CNF, this comes with a price. Such conversion can lead to an undesirable increase of the grammar size [105].

Originally, Earley's parser was designed in the field of computational linguistics as a string parsing algorithm. The parser uses a dynamic programming approach that decides whether a given one-dimensional structure (i.e. a string) belongs to a given CFG. This original formulation was improved upon by various authors over time. The work of Stolcke [170] introduced an extension to stochastic context free grammars (SCFGs). Given the stochastic nature of these grammars, the upgraded parser output also includes the *probability* that a string is produced by the given grammar.

Some authors attempted to generalize the Earley parser by considering different input domains. In [34], relation grammars were proposed to model multi-dimensional structures. Wild [200] used multiple context free grammars (MCFGs) to model RNA interaction problems. Although similar in notation, their definition of a 2D-SCFG is very different from ours, since in MCFGs non-terminals produce tuples of words of given dimension. Costagliola and Chang [33] introduced positional grammars that explicitly model the spatial relations between symbols in grammar productions. This formalism was used for parsing arithmetical expressions. The approach closest to ours was presented by Tomita [186], who introduced a straightforward extension of the Earley parser to two-dimensional grid structures. However, some cases of input examples were not handled properly in this algorithm. In Section A.4 we show a failure example where the approach of [186] does not properly parse a 2D grid, in contrast to our proposed method. Furthermore, our methods generalizes to 2D stochastic CFGs, which are much less studied in literature. One of the few existing 2D SCFG parsing approaches was used for recognition of mathematical expressions [217]; however, this approach uses a 2D CKY parser, which requires the grammar to be in Chomsky Normal Form.

A.2 2D-ASCFGs

Here we repeat the Section 7.3 as a reminder of the definition of this particular grammar type.

A two-dimensional attributed stochastic context-free grammar (2D-ASCFG) is defined as a tuple $G = (N, T, S, R, P, A)$, where N is a set of non-terminal symbols, T a set of terminal symbols, S the starting non-terminal symbol or

axiom, R a set of production rules, $\{P(r), r \in R\}$ a set of rule probabilities and $\{A(r), r \in R\}$ a set of rule attributes.

Every symbol is associated with the corresponding shape, representing a rectangular region. Starting from the axiom, production rules subdivide the starting shape either in horizontal or vertical directions. We define the set R as a union of horizontal and vertical productions: $R = R_h \cup R_v$. These productions correspond to standard horizontal and vertical split operators in split grammars. A production is of the form $X \to \lambda$, where $X \in N$ is called the left-hand-side (LHS), and $\lambda \in (N \cup T)^+$ is called the right-hand-side (RHS) of the production.

For every production, $P(X \to \lambda)$ is defined as the probability that the rule is selected in the top-down derivation from the grammar. For the grammar to be well-formed, the productions with X as LHS must satisfy the condition $\sum_\lambda P(X \to \lambda) = 1$. Additionally, each grammar rule r is associated with a set of attributes $A(r) = \{\alpha_i\}$. The elements of a single attribute are the relative sizes of the RHS shapes in respect to their parent shape, in the splitting direction: $\alpha_i = \{s_1, ..., s_{|\lambda|}\}, \sum_i s_i = 1$. These relative sizes sum up to one because RHS shapes always fill the entire shape of their parent.

We denote by τ a parse tree from the grammar, rooted on the axiom, its interior nodes corresponding to non-terminal symbols, and its exterior nodes to terminal symbols. The parse tree is obtained by applying a sequence of rules on the axiom and non-terminal nodes. A derivation from the grammar consists of the parse tree and the selected attributes at each node: $\delta = (\tau, \alpha)$. The probability of a single derivation is the product of all rule probabilities selected at each node s of the parse tree: $P(\delta) = \prod_{s \in \delta} P(r_s)$. The set of terminal nodes of a parse tree defines a lattice over an area. A lattice, or a grid is a rectangular tessellation of 2D space, exactly filling the shape of the axiom. We define the likelihood of the grammar G generating a lattice l as $L(l|G) = \sum_{\delta \Rightarrow l} P(\delta)$, where we sum over the probabilities of all derivations that yield a particular lattice.

A.3 2D Earley Parser

In this section, we mostly follow the notation from [170] and [186], adapting it as necessary. For a set of input symbols organized in a grid I of width n and height m,

$$\begin{matrix} I_{11} & I_{21} & \dots & I_{n1} \\ I_{12} & I_{22} & \dots & I_{n2} \\ \vdots & & & \vdots \\ I_{1m} & I_{2m} & \dots & I_{nm} \end{matrix}$$

the parser keeps track of a set of *states* (alternatively called *items*) for each position in the input. Each of these states corresponds to a possible derivation of the grammar that is consistent with the input up to the given point in the grid. There can be multiple states at the same point in the grid, and they are denoted as a *state set.* All of these state sets together form the *Earley chart.* Unlike 1-dimensional Earley parsing, where the chart is a one-dimensional list of state sets, 2D Earley chart is a parse table of the form

$$\begin{matrix} S_{00} & S_{10} & \dots & S_{n0} \\ S_{01} & S_{11} & \dots & S_{n1} \\ \vdots & & & \vdots \\ S_{0m} & S_{1m} & \dots & S_{nm} \end{matrix}$$

Note that the state chart contains an additional row and column in comparison to the input. These correspond to the states when no symbols have been processed in the first input row or column. Each state is defined as a tuple (p, d, o, b):

- $p \in R_H \cup R_v$ is a production of the grammar.
- The *dot position* d is an index denoting the current position in the RHS of production $p(X \to \lambda\mu)$:

$$X \to \lambda.\mu \tag{A.1}$$

where $\lambda, \mu \in (N \cup T)^+$. This indicates that the RHS of the production was expanded up to the position indicated by the dot. The dot position $d = 0$ signifies that no RHS symbols have been analyzed so far. In contrast, the dot to the right of entire RHS corresponds to a fully expanded non-terminal X. Such a state is called *complete.*

- The *origin position* o is defined a pair of indices (i, j) corresponding to the position of the state in the Earley chart.
- The *bounding box* $b(x, y, X, Y)$ represents the admissible rectangular region of the input I for the current state. For all b, the bounding box must be

non-empty : $x < X$, $y < Y$. Additionally, this region has to be contained within the input grid: $0 \leq x, X \leq n$, $0 \leq y, Y \leq m$. Finally, the origin position must always lie within the bounding box: $i < X$, $j < Y$.

We will use the following short-hand formulation to describe a state:

$$(i, j)\, X \to \lambda.\mu\, (x, y, X, Y) \tag{A.2}$$

To simplify the exposition of the algorithm, we convert the grammar into *Simple Normal Form (SNF)* [170]. In this form, terminal symbols can appear only on the right-hand side of lexical productions, i.e. productions with a single terminal symbol on the RHS. All other productions can contain an arbitrary number of non-terminal symbols. This form can be easily achieved by simply "shadowing" every terminal symbol with an additional non-terminal symbol, e.g:

$$\begin{gathered} X \to aYc \\ \Downarrow \\ A \to a \\ C \to c \\ X \to AYC \end{gathered}$$

where $X, Y, A, C \in N$; $a, c \in T$. We have chosen, without loss of generality, the convention that lexical productions are contained in the set of horizontal productions. We also add a *starting production* to the grammar, designated as $\epsilon \to S$, where ϵ is an empty symbol and S is the axiom of the grammar.

A.3.1 The Parsing Algorithm

Now we have all the prerequisites to describe the 2D Earley parsing algorithm. At the beginning, an *initial state* is created and subsequently inserted into the $(0, 0)$ cell of the Earley chart*:*

$$(0, 0)\, \epsilon \to .S\, (0, 0, n, m) \tag{A.3}$$

Afterwards, the parser performs three operations: *prediction, scanning* and *completion.*

Prediction

Given an incomplete state (the dot is not in the final position), the role of *prediction* step is to enumerate all potential expansions of the non-terminal to

the right of the dot. For each state

$$s: \quad (i,j)\, A \to \lambda.B\mu\, (x,y,X,Y) \tag{A.4}$$

the algorithm finds all possible expansions $B \to \nu$ of non-terminal B in the grammar and creates new states to be added to the chart, of the form:

$$s_{new}: \quad (i,j)\, B \to .\nu\, (i,j,X,Y) \tag{A.5}$$

Scanning

In the scanning step, we read the input symbol at the current position in the grid. All states that are in accordance with the input symbol are discovered. Note that, as the grammar is in SNF, the only states considered are the ones containing a lexical production. For each state

$$s: \quad (i,j)\, A \to .a\, (x,y,X,Y) \tag{A.6}$$

where $a \in T$ matches the input symbol $I_{i+1,j+1}$, add a new state by moving the dot over the scanned symbol:

$$s_{new}: \quad (i+1,j)\, A \to a.\, (i,j,i+1,j+1) \tag{A.7}$$

The produced states have the dot on the far right of the RHS, and are thus complete.

Completion

In the previous step, a complete state was produced from a predicted state by scanning the input. The predecessors of the predicted state now need to be updated to take into account the scanned symbol. For each complete state

$$s: \quad (i,j)\, B \to \nu.\, (x,y,X,Y) \tag{A.8}$$

find candidate states s' in Earley chart cell (x,y), which have B as the non-terminal after the dot:

$$s': \quad (x,y)\, A \to \lambda.B\mu\, (x',y',X',Y') \tag{A.9}$$

The necessary condition for a candidate state to be considered is that its bounding box encompasses the bounding box of the complete state:

$$x' \leq x \wedge y' < y \wedge X' \geq X \wedge Y' \geq Y \tag{A.10}$$

Productions $p: B \to \nu$ and $p': A \to \lambda B \mu$ belong either to the set of horizontal or vertical productions, thus there are four different combinations:

1. $p \in \mathbf{R}_h, p' \in \mathbf{R}_h$
   If $Y' = Y \vee \lambda = \emptyset$, add state $s_{new} = (i,j)\, A \rightarrow \lambda B.\mu\,(x', y', X'', Y)$, where
   $X'' = \begin{cases} X, & if\ \mu = \emptyset \\ X', & otherwise \end{cases}$

2. $p \in \mathbf{R}_v, p' \in \mathbf{R}_h$
   If $Y' = Y \vee \lambda = \emptyset$, add state $s_{new} = (X,y)\, A \rightarrow \lambda B.\mu\,(x', y', X'', Y)$, where $X'' = \begin{cases} X, & if\ \mu = \emptyset \\ X', & otherwise \end{cases}$

3. $p \in \mathbf{R}_h, p' \in \mathbf{R}_v$
   If $X' = X \vee \lambda = \emptyset$, add state $s_{new} = (x,Y)\, A \rightarrow \lambda B.\mu\,(x', y', X, Y'')$, where $Y'' = \begin{cases} Y, & if\ \mu = \emptyset \\ Y', & otherwise \end{cases}$

4. $p \in \mathbf{R}_v, p' \in \mathbf{R}_v$
   If $X' = X \vee \lambda = \emptyset$, add state $s_{new} = (i,j)\, A \rightarrow \lambda B.\mu\,(x', y', X, Y'')$, where
   $Y'' = \begin{cases} Y, & if\ \mu = \emptyset \\ Y', & otherwise \end{cases}$

These conditions simply state that the bounding box edges of states s and s' must align vertically ($Y' = Y$), or horizontally ($X' = X$), unless s' has not been expanded. The definitions of X'' and Y'' are necessary if the newly created state s_{new} is complete ($\mu = \emptyset$). In that case, the bounding box of state s' must be clipped to the value given by s (see Figure A.1). Additionally, the completion step will have to be performed on s_{new} as well.

A special case during completion occurs when symbol B is identical to the grammar axiom S, i.e. the candidate production is the starting production:

$$s: \quad (i,j)\, \epsilon \rightarrow .S\,(x', y', X', Y') \tag{A.11}$$

In that case, we add a state

$$s_{new}: \quad (i,j)\, \epsilon \rightarrow S.\,(x, y, X, Y) \tag{A.12}$$

If such a state is produced during parsing, one valid parse of the input grid was found, under the condition that the whole input grid is contained in the bounding box of the state. Otherwise, only a subset of the input was parsed. Therefore, a *final state* must satisfy the following conditions:

$$p = \epsilon \rightarrow S, \quad d = 1, \quad b = (0, 0, n, m) \tag{A.13}$$

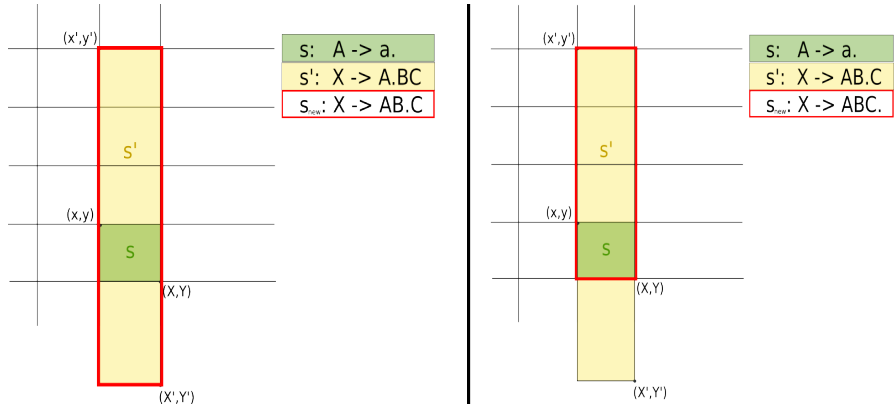


Figure A.1: Completion, case 3: (a) Bounding boxes of both complete state s and candidate state s' are aligned. The completed state s_{new} has the same bounding box as s'. (b) State s completes the expansion of non-terminal in state s', therefore limiting the bounding box of the created state s_{new}.

Note that, due to possible grammar ambiguity, there might be multiple final states. Each final state corresponds to a different parse of the input. Furthermore, there is no constraint on the position of the final state in the Earley chart. This contrasts with one-dimensional Earley parsing, where the final state is found only in the last state set of a one-dimensional state chart. Algorithm 2 provides an overview of 2D Earley parsing of an input grid with a given grammar. Functions *Completer, Predictor* and *Scanner* correspond to previously defined operations of the parser.

A.3.2 Remarks

So far, a state was uniquely defined with a tuple $s = (p, d, o, b)$. However, it is also necessary to remember the entire *scanning history* of the state, i.e. to keep track of the symbols in the input grid which were scanned by following the path to the current state. Otherwise, the algorithm may produce two "equal" states in the same Earley chart cell by having scanned different parts of the input grid. One of these states would then overwrite the other, destroying one parsing path completely! Therefore, each state is augmented with a matrix H of the same size as the input grid. This matrix is empty in the initial state.

Algorithm 2 2D Earley parsing

```
function EarleyParse2D(input, grammar)
    stateQueue ← ∅
    initialState ← [ϵ → .S (0, 0, input.n, input.m)]
    push initialState to end of stateQueue
    while stateQueue ≠ ∅ do
        pop state from front of stateQueue
        if state is complete then
            newState ← Completer(state, grammar)
        else
            Y ← state.p.rhs(d)                          ▷ Symbol after the dot.
            if Y ∈ grammar.N then
                newState ← Predictor(state, grammar)
            else if Y ∈ grammar.T then
                newState ← Scanner(state, input, grammar)
            end if
        end if
        if newState ≠ ∅ then
            push newState to end of stateQueue
        end if
    end while
end function
```

During **prediction**, the scanning history remains unchanged:

$$s_{new}.H \quad = s.H$$

In the **scanning** step, the new state inherits the history from the old state and updates it with the newly scanned symbol, under the condition that the same position in the grid has not been scanned yet:

$$s_{new}.H \quad = s.H$$

$$s_{new}.H[i][j] = a$$

However, if the same position in grid has already been scanned ($s.H[i][j] \neq \emptyset$), a new state will not be created. Finally, during **completion**, we make sure that the complete state s and the candidate state s' are "scanning history-compatible":

$$s.H[i][j] = \emptyset \vee s.H[i][j] = \emptyset \vee (s.H[i][j] = s'.H[i][j]) \quad \forall (i, j)$$

If the states are not compatible, completion does not produce a new state. Otherwise, scanning history is propagated from the complete state to the newly created state:

$$s_{new}.H \quad = s.H$$

A.3.3 An Example

We now demonstrate the algorithm on a simple grammar with 5 productions and a $2 * 2$ input grid, also used in [186]. The grammar contains the following productions:

$$p_1 : S \xrightarrow{H} AA, \quad p_2 : A \xrightarrow{V} BC, \quad p_3 : B \xrightarrow{H} b, \quad p_4 : C \xrightarrow{H} c, \quad p_5 : C \xrightarrow{H} d,$$

where we have used short-hand notations $\xrightarrow{H}$ and $\xrightarrow{V}$ for horizontal and vertical productions, respectively. We omit the scanning matrix H for simplicity. Non-terminal symbols are denoted with capital letters; conversely, terminals are shown in lowercase. S is the grammar axiom. The starting production $\epsilon \rightarrow S$ is added implicitly to the grammar. Note that the grammar is non-deterministic, since a single non-terminal C can be expanded in two different ways. We revisit stochastic productions in Section A.4.

We now show, step by step, the complete Earley chart of the parser as it processes the input sequence

$$\begin{matrix} b & b \\ c & d \end{matrix}$$

Obviously, this grid is a valid derivation from the grammar, and there is only a single way to produce it, therefore we expect a single final state. The state chart is seeded with the initial state, corresponding to the starting production.

	0	1	2
0	$\boldsymbol{\epsilon \xrightarrow{H} .S}$ (0,0,2,2)		
1			
2			

In the prediction step, there is only one state to analyse. Production p_1 is used as the only expansion of non-terminal S, and a new state is added to the same state set.

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,2) $\boldsymbol{S \xrightarrow{H} .AA}$ (0,0,2,2)		
1			
2			

Prediction continues until there are no more predictions to be made.

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,2) $S \xrightarrow{H} .AA$ (0,0,2,2) $\boldsymbol{A \xrightarrow{V} .BC}$ (0,0,2,2) $\boldsymbol{B \xrightarrow{H} .b}$ (0,0,2,2)		
1			
2			

Now, using the last state in the chart in the cell $(0, 0)$, the parser scans over the input grid on the position $(1, 1)$, i.e. symbol b. This results with a scanned and complete state being added to the appropriate position in the chart.

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,2) $S \xrightarrow{H} .AA$ (0,0,2,2) $A \xrightarrow{V} .BC$ (0,0,2,2) $B \xrightarrow{H} .b$ (0,0,2,2)	$\boldsymbol{B \xrightarrow{H} b.}$ (0,0,1,1)	
1			
2			

Note that the bounding box of the state is adjusted so that it covers only the scanned symbol. Now, the completion step is invoked, as there is a complete state in the current set. The upper-left corner of the complete state is $(0, 0)$, so we search for candidate states in that chart cell. The only state that satisfies the conditions for completion is $A \xrightarrow{V} .BC$ (0,0,2,2): it has symbol B after the dot, its bounding box encompasses the bounding box of the complete state, and the dot is in the initial position. The complete and candidate state contain horizontal and vertical productions, respectively, so we follow the rules in Section A.3.1 and add the completed state in chart cell $(0, 1)$.

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,2) $S \xrightarrow{H} .AA$ (0,0,2,2) $A \xrightarrow{H} .BC$ (0,0,2,2) $B \xrightarrow{H} .b$ (0,0,2,2)	$B \xrightarrow{H} b.$ (0,0,1,1)	
1	$\boldsymbol{A \xrightarrow{V} B.C}$ (0,0,1,1)		
2			

The newly added state passes through the prediction step and creates two predicted states, corresponding to two productions for non-terminal C.

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,2) $S \xrightarrow{H} .AA$ (0,0,2,2) $A \xrightarrow{V} .BC$ (0,0,2,2) $B \xrightarrow{H} .b$ (0,0,2,2)	$B \xrightarrow{H} b.$ (0,0,1,1)	
1	$A \xrightarrow{V} B.C$ (0,0,1,2) $\boldsymbol{C \xrightarrow{H} .c}$ (0,1,1,2) $\boldsymbol{C \xrightarrow{H} .d}$ (0,1,1,2)		
2			

The second prediction is discarded during scanning, as the terminal symbol does not match the input. The first prediction, however, creates a new scanned state.

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,2) $S \xrightarrow{H} .AA$ (0,0,2,2) $A \xrightarrow{V} .BC$ (0,0,2,2) $B \xrightarrow{H} .b$ (0,0,2,2)	$B \xrightarrow{H} b.$ (0,0,1,1)	
1	$A \xrightarrow{V} B.C$ (0,0,1,2) $C \xrightarrow{H} .c$ (0,1,1,2) $C \xrightarrow{H} .d$ (0,1,1,2)	$\boldsymbol{C \xrightarrow{H} c.}$ (0,1,1,2)	
2			

The completion step now finishes the derivation of non-terminal A in cell $(0, 2)$.

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,2) $S \xrightarrow{H} .AA$ (0,0,2,2) $A \xrightarrow{V} .BC$ (0,0,2,2) $B \xrightarrow{H} .b$ (0,0,2,2)	$B \xrightarrow{H} b.$ (0,0,1,1)	
1	$A \xrightarrow{V} B.C$ (0,0,1,2) $C \xrightarrow{H} .c$ (0,1,1,2) $C \xrightarrow{H} .d$ (0,1,1,2)	$C \xrightarrow{H} c.$ (0,1,1,2)	
2	$\boldsymbol{A \xrightarrow{V} BC.}$ (0,0,1,2)		

Note that the bounding box of the completed state in cell $(2, 0)$ corresponds to scanning the first column of the input grid. This completed state in turn triggers a completion, resulting with the second state in cell $(1, 0)$.

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,2) $S \xrightarrow{H} .AA$ (0,0,2,2) $A \xrightarrow{V} .BC$ (0,0,2,2) $B \xrightarrow{H} .b$ (0,0,2,2)	$B \xrightarrow{H} b.$ (0,0,1,1) $\boldsymbol{S \xrightarrow{H} A.A}$ (0,0,2,2)	
1	$A \xrightarrow{V} B.C$ (0,0,1,2) $C \xrightarrow{H} .c$ (0,1,1,2) $C \xrightarrow{H} .d$ (0,1,1,2)	$C \xrightarrow{H} c.$ (0,1,1,2)	
2	$A \xrightarrow{V} BC.$ (0,0,1,2)		

The bounding box of this newest state still encompasses the whole input grid, as there are more non-terminals on the production right-hand side that have not been expanded. Next, this state is used for prediction, creating two new states, one of which is immediately used in the scanning step.

	0	1	2
0	$\epsilon \overset{H}{\rightarrow} .S$ (0,0,2,2) $S \overset{H}{\rightarrow} .AA$ (0,0,2,2) $A \overset{V}{\rightarrow} .BC$ (0,0,2,2) $B \overset{H}{\rightarrow} .b$ (0,0,2,2)	$B \overset{H}{\rightarrow} b.$ (0,0,1,1) $S \overset{H}{\rightarrow} A.A$ (0,0,2,2) $\boldsymbol{A \overset{V}{\rightarrow} .BC}$ (1,0,2,2) $\boldsymbol{B \overset{H}{\rightarrow} .b}$ (1,0,2,2)	$\boldsymbol{B \overset{H}{\rightarrow} b.}$ (1,0,2,1)
1	$A \overset{V}{\rightarrow} B.C$ (0,0,1,2) $C \overset{H}{\rightarrow} .c$ (0,1,1,2) $C \overset{H}{\rightarrow} .d$ (0,1,1,2)	$C \overset{H}{\rightarrow} c.$ (0,1,1,2)	
2	$A \overset{V}{\rightarrow} BC.$ (0,0,1,2)		

After the completion and prediction steps:

	0	1	2
0	$\epsilon \overset{H}{\rightarrow} .S$ (0,0,2,2) $S \overset{H}{\rightarrow} .AA$ (0,0,2,2) $A \overset{V}{\rightarrow} .BC$ (0,0,2,2) $B \overset{H}{\rightarrow} .b$ (0,0,2,2)	$B \overset{H}{\rightarrow} b.$ (0,0,1,1) $S \overset{H}{\rightarrow} A.A$ (0,0,2,2) $A \overset{V}{\rightarrow} .BC$ (1,0,2,2) $B \overset{H}{\rightarrow} .b$ (1,0,2,2)	$B \overset{H}{\rightarrow} b.$ (1,0,2,1)
1	$A \overset{V}{\rightarrow} B.C$ (0,0,1,2) $C \overset{H}{\rightarrow} .c$ (0,1,1,2) $C \overset{H}{\rightarrow} .d$ (0,1,1,2)	$C \overset{H}{\rightarrow} c.$ (0,1,1,2) $\boldsymbol{A \overset{V}{\rightarrow} B.C}$ (1,0,2,2) $\boldsymbol{C \overset{H}{\rightarrow} .c}$ (1,1,2,2) $\boldsymbol{C \overset{H}{\rightarrow} .d}$ (1,1,2,2)	
2	$A \overset{V}{\rightarrow} BC.$ (0,0,1,2)		

The last symbol in the grid is subsequently scanned, which triggers a recursive completion step, resulting with a final state.

	0	1	2
0	$\epsilon \overset{H}{\rightarrow} .S$ (0,0,2,2) $S \overset{H}{\rightarrow} .AA$ (0,0,2,2) $A \overset{V}{\rightarrow} .BC$ (0,0,2,2) $B \overset{H}{\rightarrow} .b$ (0,0,2,2)	$B \overset{H}{\rightarrow} b.$ (0,0,1,1) $S \overset{H}{\rightarrow} A.A$ (0,0,2,2) $A \overset{V}{\rightarrow} .BC$ (1,0,2,2) $B \overset{H}{\rightarrow} .b$ (1,0,2,2)	$B \overset{H}{\rightarrow} b.$ (1,0,2,1) $\boldsymbol{S \overset{H}{\rightarrow} AA.}$ (0,0,2,2) $\underline{\epsilon \overset{\mathbf{H}}{\rightarrow} \mathbf{S}.}$ (0,0,2,2)
1	$A \overset{V}{\rightarrow} B.C$ (0,0,1,2) $C \overset{H}{\rightarrow} .c$ (0,1,1,2) $C \overset{H}{\rightarrow} .d$ (0,1,1,2)	$C \overset{H}{\rightarrow} c.$ (0,1,1,2) $A \overset{V}{\rightarrow} B.C$ (1,0,2,2) $C \overset{H}{\rightarrow} .c$ (1,1,2,2) $C \overset{H}{\rightarrow} .d$ (1,1,2,2)	$\boldsymbol{C \overset{H}{\rightarrow} d.}$ (1,1,2,2)
2	$A \overset{V}{\rightarrow} BC.$ (0,0,1,2)	$\boldsymbol{A \overset{V}{\rightarrow} BC.}$ (1,0,2,2)	

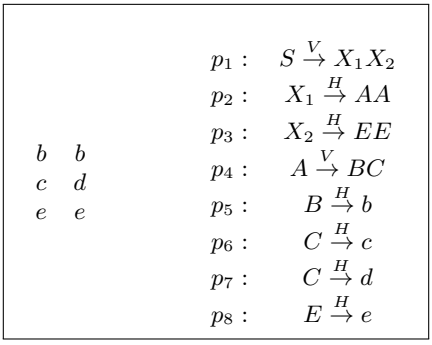


Figure A.2: A more difficult example. Left: Input grid, right: input grammar

The underlined state satisfies all final state conditions, which means that the algorithm has found a valid parse of the input. In contrast, by following the method described in [186], the final parse chart would be as follows:

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,2) $S \xrightarrow{H} .AA$ (0,0,2,2) $A \xrightarrow{V} .BC$ (0,0,2,2) $B \xrightarrow{H} .b$ (0,0,2,2)	$B \xrightarrow{H} b.$ (0,0,2,1)	$S \xrightarrow{H} A.A$ (0,0,2,2) $A \xrightarrow{V} .BC$ (2,0,2,2) $B \xrightarrow{H} .b$ (2,0,2,2)
1	$A \xrightarrow{V} B.C$ (0,0,2,2) $C \xrightarrow{H} .c$ (0,1,2,2) $C \xrightarrow{H} .d$ (0,1,2,2)	$C \xrightarrow{H} c.$ (0,1,2,2)	
2	$A \xrightarrow{V} BC.$ (0,0,2,2)		

Notice the different bounding box of non-terminal B in chart position $(1, 0)$ which propagates to subsequent states, making it impossible to parse the input. Since Figure 18.3 in [186] shows a successful parse of this input, we can only conclude there is an omission in their algorithm, likely in the scanning step. However, even if the scanning step of their approach is fixed to calculate the correct bounding box, the algorithm of [186] does not properly handle cases such as the one exemplified in Figure A.1. We demonstrate this on a more challenging input, shown in Figure A.2.

We parse the input with both the approach of [186] (after fixing the scanning step) and our approach, and show the final Earley charts in Figure A.3. Our approach successfully parses the input with the given grammar, while the

approach of [186] fails due to absence of X'' and Y'' terms in the completion step.

A.4 Extension to Stochastic Grammars

As said in Section A.2, in a stochastic context free grammar (SCFG), every production contains an associated probability. Consequently, the grammar generates each output derivation with an associated probability. Due to ambiguities in the grammar, there might be multiple ways of generating a single output. As defined previously, the likelihood of a grammar G generating a lattice l is defined as $L(l|G) = \sum_{\delta \Rightarrow l} P(\delta)$, summed over all derivations that yield a particular lattice (grid). However, among these alternative derivations, we are interested in the *most-likely path,* i.e. the sequence of rules associated with the maximum probability. This sequence is called the *Viterbi parse,* and its corresponding probability the *Viterbi probability.*

Our 2D Earley parser can be easily adapted to calculate the Viterbi parse of a given derivation, by following the procedure described in [170]. Essentially, every state in the Earley chart is assigned a Viterbi probability. These probabilities are then propagated and updated during parsing. Finally, the derivation probability is determined by reading out the Viterbi probabilities in all final states in the chart, and selecting the maximum value.

To describe the probability calculation algorithm, the concept of *predecessor* states needs to be introduced. As described in Section A.3, new states are created and added to the chart based on previous states. In prediction and scanning, there is only one predecessor state, while in completion step the new state is created based on two predecessor states. Therefore, for each state, we can keep track of its predecessor(s).

Calculation of Viterbi probabilities is performed as follows. First, the initial state is assigned the Viterbi probability of 1. Subsequently, when a new state is being added to the chart, its probability is calculated depending on the parser operation:

Prediction

For a predecessor state s with Viterbi probability of p:

$$(i, j)\, A \rightarrow \lambda.B\mu \quad [p]$$

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,3) $S \xrightarrow{V} .X_1X_2$ (0,0,2,3) $X_1 \xrightarrow{H} .AA$ (0,0,2,3) $A \xrightarrow{V} .BC$ (0,0,2,3) $B \xrightarrow{H} .b$ (0,0,2,3)	$B \xrightarrow{H} b.$ (0,0,1,1) $X_1 \xrightarrow{H} A.A$ (0,0,2,3) $A \xrightarrow{V} .BC$ (1,0,2,3) $B \xrightarrow{H} .b$ (1,0,2,3)	$B \xrightarrow{H} b.$ (1,0,2,1) $X_1 \xrightarrow{H} AA.$ (0,0,2,3)
1	$A \xrightarrow{V} B.C$ (0,0,1,3) $C \xrightarrow{H} .c$ (0,1,1,3) $C \xrightarrow{H} .d$ (0,1,1,3)	$C \xrightarrow{H} c.$ (0,1,1,2) $A \xrightarrow{V} B.C$ (1,0,2,3) $C \xrightarrow{H} .c$ (1,1,2,3) $C \xrightarrow{H} .d$ (1,1,2,3)	$C \xrightarrow{H} d.$ (1,1,2,2)
2	$A \xrightarrow{V} BC.$ (0,0,1,3)	$A \xrightarrow{V} BC.$ (1,0,2,3)	
3	$S \xrightarrow{V} X_1.X_2$ (0,0,2,3) $X_2 \xrightarrow{H} .EE$ (0,3,2,3) $E \xrightarrow{H} .e$ (0,3,2,3)		

(a)

	0	1	2
0	$\epsilon \xrightarrow{H} .S$ (0,0,2,3) $S \xrightarrow{V} .X_1X_2$ (0,0,2,3) $X_1 \xrightarrow{H} .AA$ (0,0,2,3) $A \xrightarrow{V} .BC$ (0,0,2,3) $B \xrightarrow{H} .b$ (0,0,2,3)	$B \xrightarrow{H} b.$ (0,0,1,1) $X_1 \xrightarrow{H} A.A$ (0,0,2,2) $A \xrightarrow{V} .BC$ (1,0,2,2) $B \xrightarrow{H} .b$ (1,0,2,2)	$B \xrightarrow{H} b.$ (1,0,2,1) $X_1 \xrightarrow{H} AA.$ (0,0,2,2)
1	$A \xrightarrow{V} B.C$ (0,0,1,3) $C \xrightarrow{H} .c$ (0,1,1,3) $C \xrightarrow{H} .d$ (0,1,1,3)	$C \xrightarrow{H} c.$ (0,1,1,2) $A \xrightarrow{V} B.C$ (1,0,2,2) $C \xrightarrow{H} .c$ (1,1,2,2) $C \xrightarrow{H} .d$ (1,1,2,2)	$C \xrightarrow{H} d.$ (1,1,2,2)
2	$A \xrightarrow{V} BC.$ (0,0,1,2) $S \xrightarrow{V} X_1.X_2$ (0,0,2,3) $X_2 \xrightarrow{H} .EE$ (0,2,2,3) $E \xrightarrow{H} .e$ (0,2,2,3)	$A \xrightarrow{V} BC.$ (1,0,2,2) $E \xrightarrow{H} e.$ (0,2,1,3) $X_2 \xrightarrow{H} E.E$ (0,2,2,3) $E \xrightarrow{H} .e$ (1,2,2,3)	$E \xrightarrow{H} e.$ (1,2,2,3) $X_2 \xrightarrow{H} EE.$ (0,2,2,3)
3	$S \xrightarrow{V} X_1X_2.$ (0,0,2,3) $\underline{\boldsymbol{\epsilon \xrightarrow{H} S.}}$ (0,0,2,3)		

(b)

Figure A.3: Parsing the example from Fig. A.2: a) Parsing chart of [186] with fixed scanning step. No final states exist in the chart. b) Our approach. The input is correctly parsed.

the Viterbi probability of every successor state s' depends solely on the probability of the production in state s:

$$(i, j)\, B \to .\nu \quad [P(A \to \lambda B \mu)]$$

Scanning

Since the terminal symbol was already selected during prediction, Viterbi probability simply propagates to the new state. The predecessor state s:

$$(i, j)\, A \to .a \quad [p]$$

results in the successor state s':

$$(i + 1, j)\, A \to a. \quad [p]$$

Completion

Every state s'' produced by completion will have two predecessors s and s':

$$(i, j)\, B \to \nu. \quad [p]$$

$$(x, y)\, A \to \lambda . B \mu \quad [p']$$

Its Viterbi probability is then calculated by multiplying the probabilities of predecessors:

$$(i, j)\, A \to \lambda B . \mu \quad [pp']$$

However, the same state (and with the same scanning history) might be generated multiple times, because there can be several plausible input parses. Therefore, if a state already exists in a chart, its Viterbi probability and predecessor list must be updated. Since we are interested only in the parsing path of maximum probability, we replace the existing state with the new state only if the new Viterbi probability exceeds the current highest value.

A.4.1 Parser Output

After the parsing has finished, we search the Earley chart for final states. If none are found, the input cannot be parsed with the current grammar. Otherwise, we select the final state with the highest Viterbi probability, and return that value as the probability of the input grid.

Often, one also needs to know the exact productions of the grammar used to parse the input, and their count. These *expected usage counts* can be used, for example, to find the maximum likelihood production probabilities based on a set of training instances [114, 170], using an expectation-maximization algorithm. The productions used in the Viterbi parse of the input can be extracted from the Earley chart, by tracing back the path from the final state to the initial state. The outline of the procedure is shown in Algorithm 3.

Algorithm 3 Calculating expected production counts by backtracking in Earley chart

```
function BACKTRACK(state, pCounts, grammar)
    if state.d = 0 then                          ▷ State generated by prediction.
        pCounts[state.p] ← pCounts[state.p] + 1
    else
        if state.p.rhs(d) ∈ grammar.T then       ▷ State generated by scanning.
            pCounts ← BACKTRACK(state.predecessor, pCounts, grammar)
        else                                     ▷ State generated by completion.
            pCounts ← BACKTRACK(state.predecessor, pCounts, grammar)
            pCounts ← BACKTRACK(state.predecessor2, pCounts, grammar)
        end if
    end if
    return pCounts
end function
```

The recursive function *Backtrack* is called with the final state and production counts initialized to zero. The algorithm is essentially a depth-first search, since states generated by completion provide two separate paths to follow. The recursion stops whenever a state generated by prediction is encountered. At this point, the usage count of the corresponding production is incremented. When the algorithm finishes, the variable *pCounts* will contain the desired results. For instance, the output for example in Figure A.2 is [1 1 1 2 2 1 1 2].

A.5 Conclusion

We have presented a detailed description of an Earley parsing algorithm for two-dimensional stochastic context free grammars. To the best of our knowledge, this is the first algorithm that combines the insights from both deterministic multidimensional grammars parsing and one-dimensional stochastic grammar parsing in a top-down approach. Additionally, we have demonstrated our approach on two toy examples of two-dimensional grid parsing. Finally, this algorithm was successfully used in the field of inverse procedural modeling, for the task of parsing building facades, see 7.

Bibliography

[1] Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. SLIC Superpixels. Tech. rep., EPFL, EPFL, 2010.

[2] Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S. M., and Szeliski, R. Building rome in a day. *Commun. ACM 54*, 10 (Oct. 2011), 105–112.

[3] Alegre, O., and Dellaert, F. A probabilistic approach to the semantic interpretation of building facades. In *Workshop on Vision Techniques Applied to the Rehabilitation of City Centres* (2004), pp. 1–12.

[4] Aliaga, D. G., Rosen, P. A., and Bekins, D. R. Style grammars for interactive visualization of architecture. *IEEE Transactions on Visualization and Computer Graphics 13*, 4 (2007), 786–797.

[5] Anguelov, D., Taskar, B., Chatalbashev, V., Koller, D., Gupta, D., Heitz, G., and Ng, A. Discriminative learning of markov random fields for segmentation of 3d scan data. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2005).

[6] Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 33*, 5 (May 2011), 898–916.

[7] Bao, F., Schwarz, M., and Wonka, P. Procedural facade variations from a single layout. *ACM Transactions on Graphics 32*, 1 (2013).

[8] Barinova, O., Lempitsky, V., and Kohli, P. On the detection of multiple object instances using Hough transforms. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010).

[9] Becker, S., Peter, M., and Fritsch, D. Grammar-supported 3d indoor reconstruction from point clouds for as-built bim. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences II-3/W4* (2015), 17–24.

[10] Benenson, R., Mathias, M., Timofte, R., and Van Gool, L. Pedestrian detection at 100 frames per second. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012).

[11] Benenson, R., Mathias, M., Tuytelaars, T., and Van Gool, L. Seeking the strongest rigid detector. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2013).

[12] Berg, A., Grabler, F., and Malik, J. Parsing images of architectural scenes. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)* (2007).

[13] Biljecki, F., Ledoux, H., and Stoter, J. An improved LOD specification for 3D building models and its CityGML realisation with the Random3Dcity procedural modelling engine. *Computers, Environment and Urban Systems In submission* (2015).

[14] Bodis-Szomoru, A., Riemenschneider, H., and Van Gool, L. Fast, Approximate Piecewise-Planar Modeling Based on Sparse Structure-from-Motion and Superpixels. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014).

[15] Bodis-Szomoru, A., Riemenschneider, H., and Van Gool, L. Superpixel Meshes for Fast Edge-Preserving Surface Reconstruction. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[16] Boiman, O., Shechtman, E., and Irani, M. In defense of nearest-neighbor based image classification. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2008).

[17] Bokeloh, M., Wand, M., and Seidel, H.-P. A connection between partial symmetry and inverse procedural modeling. *Proceedings of the International Conference on Computer graphics and interactive techniques (SIGGRAPH) 29*, 4 (2010).

[18] Bosch, A., Zisserman, A., and Muoz, X. Scene classification using a hybrid generative/discriminative approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 30*, 4 (2008), 712–727.

[19] Boulch, A., Houllier, S., Marlet, R., and Tournaire, O. Semantizing complex 3d scenes using constrained attribute grammars. *Computer Graphics Forum 32*, 5 (2013), 33–42.

[20] Boykov, Y., and Kolmogorov, V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 26*, 9 (2004), 124–1137.

[21] Boykov, Y., Veksler, O., and Zabih, R. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 23*, 11 (2001), 1222–1239.

[22] Bronstein, A. M., Bronstein, M. M., Guibas, L. J., and Ovsjanikov, M. Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Transactions on Graphics* (2011).

[23] Brostow, G., Shotton, J., Fauqueur, J., and Cipolla, R. Segmentation and recognition using structure from motion point clouds. In *Proceedings of European Conference on Computer Vision (ECCV)* (2008).

[24] Capturing Reality s.r.o. RealityCapture. http://www.capturingreality.com/, 2014.

[25] Chang, C.-C., and Lin, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology 2* (2011), 27:1–27:27.

[26] Chib, S., and Greenberg, E. Understanding the Metropolis-Hastings Algorithm. *The American Statistician 49*, 4 (1995), 327–335.

[27] Chomsky, N. Three models for the description of language. *IRE Transactions on Information Theory 2* (1956), 113–124.

[28] Chomsky, N. *Syntactic Structures*. Mouton & Co., The Hague, 1957.

[29] Chomsky, N. Formal properties of grammars. *Handbook of Mathematical Psychology* (1963).

[30] Cohen, A., Schwing, A. G., and Pollefeys, M. Efficient structured parsing of facades using dynamic programming. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014).

[31] Comaniciu, D., and Meer, P. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 24*, 5 (2002), 603–619.

[32] Cornelis, N., Leibe, B., Cornelis, K., and Van Gool, L. J. 3d urban scene modeling integrating recognition and reconstruction. *International Journal of Computer Vision (IJCV) 78*, 2-3 (2008), 121–141.

[33] Costagliola, G., and Chang, S.-K. Using linear positional grammars for the LR parsing of 2-d symbolic languages. *Grammars 2*, 1 (1999), 1–34.

[34] Crimi, C., Guercio, A., Nota, G., Pacini, G., Tortora, G., and Tucci, M. Relation grammars and their application to multi-dimensional languages. *Journal of Visual Languages & Computing 2*, 4 (Dec. 1991), 333–346.

[35] Dai, D., Prasad, M., Schmitt, G., and Van Gool, L. Learning domain knowledge for facade labeling. In *Proceedings of European Conference on Computer Vision (ECCV)* (2012).

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

[37] Deep, K., Singh, K. P., Kansal, M., and Mohan, C. A real coded genetic algorithm for solving integer and mixed integer optimization problems. *Applied Mathematics and Computation 212*, 2 (2009), 505 – 518.

[38] Demuth, H., Beale, M., Demuth, H., and Beale, M. Neural network toolbox for use with matlab, 1993.

[39] Dick, A. R., Torr, P. H. S., and Cipolla, R. Modelling and interpretation of architecture from several images. *International Journal of Computer Vision (IJCV) 60* (2004), 111–134.

[40] Dollar, P., Tu, Z., Perona, P., and Belongie, S. Integral channel features. In *Proceedings of British Machine Vision Conference (BMVC)* (2009).

[41] Earley, J. An efficient context-free parsing algorithm. *Communications of the ACM 13*, 2 (Feb. 1970).

[42] Esri. CityEngine. http://www.esri.com/software/cityengine, 2013.

[43] Esri. Marseille Urban Planning Project. http://www.esri.com/software/cityengine/industries/marseille, 2015.

[44] Esri. Swiss Village for Masdar City. http://www.esri.com/software/cityengine/industries/swiss-village, 2015.

[45] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The PASCAL visual object classes (VOC) challenge. *International Journal of Computer Vision (IJCV) 88*, 2 (June 2010), 303–338.

[46] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 32* (2010), 1627–1645.

[47] Felzenszwalb, P. F., Girshick, R. B., and McAllester, D. A. Cascade object detection with deformable part models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010), pp. 2241–2248.

[48] Felzenszwalb, P. F., and Huttenlocher, D. P. Efficient graph-based image segmentation. *International Journal of Computer Vision (IJCV) 59* (2004), 2004.

[49] Fischler, M. A., and Bolles, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24* (June 1981), 381–395.

[50] Fletcher, R. *Practical Methods of Optimization; (2Nd Ed.)*. Wiley-Interscience, New York, NY, USA, 1987.

[51] Floros, G., Rematas, K., and Leibe, B. Multi-class image labeling with top-down segmentation and generalized robust P^N potentials. In *Proceedings of British Machine Vision Conference (BMVC)* (2011), pp. 1–11.

[52] Frey, B., and Dueck, D. Clustering by passing messages between data points. *Science 315* (2007), 972–976.

[53] Fröhlich, B., Rodner, E., and Denzler, J. Semantic segmentation with millions of features: Integrating multiple cues in a combined random forest approach. In *Proceedings of Asian Conference on Computer Vision (ACCV)* (2012), pp. 218–231.

[54] Fröhlich, B., Rodner, E., Kemmler, M., and Denzler, J. Large-scale gaussian process multi-class classification for semantic segmentation and facade recognition. *Machine Vision and Applications (MVA) 24*, 5 (2013), 1043–1053.

[55] Furukawa, Y., and Ponce, J. Accurate, Dense, and Robust Multi-View Stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 32*, 8 (2010), 1362–1376.

[56] Gadde, R. Paris art deco facades dataset. https://github.com/raghudeep/ParisArtDecoFacadesDataset, 2015.

[57] Gadde, R., Marlet, R., and Paragios, N. Learning grammars for architecture-specific facade parsing. Tech. rep., 2014.

[58] Gallup, D., Frahm, J.-M., and Pollefeys, M. Piecewise planar and non-planar stereo for urban scene reconstruction. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010), pp. 1418–1425.

[59] Geiger, A., Lauer, M., and Urtasun, R. A generative model for 3d urban scene understanding from movable platforms. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2011).

[60] Georgoulis, S., Proesmans, M., and Van Gool, L. Tackling shapes and brdfs head-on. In *2nd International Conference on 3D Vision, 3DV 2014, Tokyo, Japan, December 8-11, 2014* (2014), pp. 267–274.

[61] Gilks, W., Richardson, S., and Spiegelhalter, D. *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC Interdisciplinary Statistics. Taylor & Francis, 1995.

[62] Gips, J. Computer implementation of shape grammars. In *Workshop on Shape Computation* (1999), Massachusetts Institute of Technology.

[63] Girshick, R. B., Felzenszwalb, P. F., and McAllester, D. Discriminatively trained deformable part models, release 5. http://people.cs.uchicago.edu/ rbg/latent-release5/, 2012.

[64] Gold, E. M. Language identification in the limit. *Information and Control 10*, 5 (1967).

[65] Golovinskiy, A., Kim, V., and Funkhouser, T. Shape-based recognition of 3d point clouds in urban environments. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)* (2009).

[66] Gould, S., Fulton, R., and Koller, D. Decomposing a scene into geometric and semantically consistent regions. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)* (2009), pp. 1–8.

[67] Gould, S., Russakovsky, O., Goodfellow, I., Baumstarck, P., Ng, A. Y., and Koller, D. The stair vision library (v2.2), 2009.

[68] Grant, M., and Boyd, S. CVX: Matlab software for disciplined convex programming. http://cvxr.com/cvx, Mar. 2014.

[69] Green, P. J. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika 82*, 4 (1995).

[70] Green, P. J., and Hastie, D. I. Reversible jump mcmc. http://www.stats.bris.ac.uk/~mapjg/papers/rjmcmc_20090613.pdf, 2009.

[71] Gröger, G., Kolbe, T. H., Czerwinski, A., and Nagel, C. OpenGIS City Geography Markup Language (CityGML) Encoding Standard (OGC 08-007r1). Tech. rep., Aug. 2008.

[72] Grompone von Gioi, R., Jakubowicz, J., Morel, J.-M., and Randall, G. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 32* (2010), 722–732.

[73] Grune, D., and Jacobs, C. J. H. *Parsing Techniques: A Practical Guide*. Ellis Horwood, Upper Saddle River, NJ, USA, 1990.

[74] Han, F., and Zhu, S.-C. Bottom-up/top-down image parsing with attribute grammar. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 31*, 1 (2009).

[75] Han, T., Liu, C., Tai, C.-L., and Quan, L. Quasi-regular facade structure extraction. In *Proceedings of Asian Conference on Computer Vision (ACCV)* (2012), pp. 552–564.

[76] Havemann, S. *Generative mesh modeling*. PhD thesis, University of Braunschweig - Institute of Technology, 2005.

[77] Heitz, G., and Koller, D. Learning spatial context: Using stuff to find things. In *Proceedings of European Conference on Computer Vision (ECCV)* (2008), pp. 30–43.

[78] Holland, J. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[79] Horning, J. J. *A study of grammatical inference*. PhD thesis, Stanford University, 1969. AAI7010465.

[80] Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., and Igel, C. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks* (2013).

[81] Huang, Q., Koltun, V., and Guibas, L. Joint shape segmentation with linear programming. *ACMGraphics 30*, 6 (2011), 125:1–125:12.

[82] Hwang, I., Stuhlmüller, A., and Goodman, N. D. Inducing probabilistic programs by bayesian program merging. *CoRR arXiv:1110.5667* (2011).

[83] Jancosek, M., and Pajdla, T. Multi-View Reconstruction Preserving Weakly-Supported Surfaces. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2011).

[84] Johnson, A., and Hebert, M. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (1999).

[85] Kaehler, O., and Reid, I. Efficient 3D Scene Labeling Using Fields of Trees. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)* (2013).

[86] Kalogerakis, E., Hertzmann, A., and Singh, K. Learning 3D Mesh Segmentation and Labeling. *ACM Transactions on Graphics 29*, 3 (2010).

[87] Karp, R. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

[88] Kim, B., Kohli, P., and Savarese, S. 3D Scene Understanding by Voxel-CRF. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)* (2013).

[89] Kimberly, D., Frischer, B., Müller, P., Ulmer, A., and Haegler, S. Rome reborn 2.0: A case study of virtual city reconstruction using procedural modeling techniques. In *Proceedings of the 37th international conference on computer applications and quantitative methods in archaeology (CAA)* (2009), pp. 62–66.

[90] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science 220* (1983), 671–680.

[91] Knopp, J., Prasad, M., and Van Gool, L. Scene cut: Class-specific object detection and segmentation in 3d scenes. In *3DIMPVT* (2011).

[92] Kolmogorov, V., and Zabih, R. What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 26* (2004), 65–81.

[93] Korah, T., and Rasmussen, C. Analysis of building textures for reconstructing partially occluded facades. In *Proceedings of European Conference on Computer Vision (ECCV)* (2008), pp. 359–372.

[94] Korč, F., and Förstner, W. eTRIMS Image Database for interpreting images of man-made scenes. Tech. Rep. TR-IGG-P-2009-01, Dept. of Photogrammetry, University of Bonn, April 2009.

[95] Koutsourakis, P., Simon, L., Teboul, O., Tziritas, G., and Paragios, N. Single view reconstruction using shape grammars for urban environments. In *ICCV* (2009), IEEE, pp. 1795–1802.

[96] Kozinski, M., Gadde, R., Zagoruyko, S., Marlet, R., and Obozinski, G. 2d adjacency patterns for accurate image parsing with occlusions. In *CVPR* (2015).

[97] Kozinski, M., and Marlet, R. Image Parsing with Graph Grammars and Markov Random Fields Applied to Facade Analysis. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)* (2014).

[98] Krecklau, L., and Kobbelt, L. Procedural modeling of interconnected structures. *Computer Graphics Forum 30*, 2 (2011), 335–344.

[99] Krecklau, L., Pavic, D., and Kobbelt, L. Generalized use of non-terminal symbols for procedural modeling. *Computer Graphics Forum 29*, 8 (2010), 2291–2303.

[100] Kumar, M. P., and Koller, D. Efficiently selecting regions for scene understanding. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010), pp. 3217–3224.

[101] Kumar, S., August, J., and Hebert, M. Exploiting inference for approximate parameter learning in discriminative fields: an empirical study. In *Proceedings of the 5th international conference on Energy Minimization Methods in Computer Vision and Pattern Recognition* (Berlin, Heidelberg, 2005), Springer-Verlag, pp. 153–168.

[102] Ladicky, L., Sturgess, P., Alahari, K., Russell, C., and Torr, P. H. S. What, where and how many? combining object detectors and crfs. In *Proceedings of European Conference on Computer Vision (ECCV)* (2010), pp. 424–437.

[103] Ladicky, L., Sturgess, P., Russell, C., Sengupta, S., Bastanlar, Y., Clocksin, W., and Torr, P. Joint Optimization for Object Class Segmentation and Dense Stereo Reconstruction. *International Journal of Computer Vision (IJCV) 100*, 2 (2012), 122–133.

[104] Lafarge, F., Descombes, X., Zerubia, J., and Pierrot Deseilligny, M. Structural approach for building reconstruction from a single DSM. *ACM Transactions on Graphics 32*, 1 (2010), 135–147.

[105] Lange, M., and Leiss, H. To CNF or not to CNF? An efficient yet presentable version of the CYK algorithm. *Informatica Didactica 8* (2009).

[106] Lefebvre, S., Hornus, S., and Lasram, A. By-example synthesis of architectural textures. *SIGGRAPH 29*, 4 (July 2010), 84:1–84:8.

[107] Leibe, B., Cornelis, N., Cornelis, K., and Van Gool, L. J. Dynamic 3d scene analysis from a moving vehicle. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2007).

[108] Leibe, B., Leonardis, A., and Schiele, B. An implicit shape model for combined object categorization and segmentation. In *Toward Category-Level Object Recognition* (2006), pp. 508–524.

[109] Liebowitz, D., and Zisserman, A. Metric rectification for perspective images of planes. In *IEEE Conference on Computer Vision and Pattern Recognition* (1998), pp. 482–488.

[110] Lin, J., Cohen-Or, D., Zhang, H. R., Liang, C., Sharf, A., Deussen, O., and Chen, B. Structure-preserving retargeting of irregular 3d architecture. *ACM Transactions on Graphics 30*, 6 (2011).

[111] Liu, C., Yuen, J., and Torralba, A. Nonparametric scene parsing via label transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*, 12 (2011), 2368–2382.

[112] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV) 60*, 2 (Nov. 2004), 91.

[113] Martinović, A., Mathias, M., Weissenberg, J., and Van Gool, L. A three-layered approach to facade parsing. In *ECCV* (2012).

[114] Martinović, A., and Van Gool, L. Bayesian grammar learning for inverse procedural modeling. In *CVPR* (2013).

[115] Martinović, A., and Van Gool, L. Hierarchical co-segmentation of building facades. In *V3D* (2014).

[116] Mathias, M., Martinović, A., Weissenberg, J., Haegler, S., and Van Gool, L. Automatic architectural style recognition. In *ARCH3D* (2011).

[117] Mathias, M., Martinović, A., Weissenberg, J., and Van Gool, L. Procedural 3d building reconstruction using shape grammars and detectors. In *IEEE International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)* (2011), pp. 304–311.

[118] Mathias, M., Timofte, R., Benenson, R., and Van Gool, L. Traffic sign recognition - how far are we from the solution? In *International Joint Conference on Neural Networks* (2013).

[119] Mathworks. *Global Optimization Toolbox Documentation.* Natick, Massachusetts, 2014.

[120] Mattausch, O., Panozzo, D., Mura, C., Sorkine-Hornung, O., and Pajarola, R. Object detection and classification from large-scale cluttered indoor scans. *Proceedings of Eurographics (EUROGRAPHICS)* (2014).

[121] Mikolajczyk, K., and Schmid, C. Scale & affine invariant interest point detectors. *International Journal of Computer Vision (IJCV) 60* (October 2004), 63–86.

[122] Muja, M., and Lowe, D. G. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application (VISSAPP)* (2009), INSTICC Press, pp. 331–340.

[123] Müller, P., Vereenooghe, T., Ulmer, A., and Van Gool, L. Automatic reconstruction of roman housing architecture. In *Recording, Modeling and Visualization of Cultural Heritage* (2005), pp. 287–298.

[124] Müller, P., Vereenooghe, T., Wonka, P., Paap, I., and Van Gool, L. Procedural 3d reconstruction of puuc buildings in xkipché. In *Eurographics Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)* (2006), EG, pp. 139–146.

[125] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. Procedural modeling of buildings. In *SIGGRAPH* (2006).

[126] Müller, P., Zeng, G., Wonka, P., and Van Gool, L. Image-based procedural modeling of facades. *ACMGraphics 26*, 3 (2007), 85.

[127] Munoz, D., Bagnell, J., and Hebert, M. Co-inference for Multi-modal Scene Analysis. In *Proceedings of European Conference on Computer Vision (ECCV)* (2012).

[128] Musialski, P., Wimmer, M., and Wonka, P. Interactive coherence-based facade modeling. *Computer Graphics Forum 31*, 2pt3 (2012), 661–670.

[129] Musialski, P., Wonka, P., Aliaga, D. G., Wimmer, M., Van Gool, L., and Purgathofer, W. A survey of urban reconstruction. In *EUROGRAPHICS 2012 State of the Art Reports* (May 2012), EG STARs, Eurographics Association, pp. 1–28.

[130] Nan, L., Xie, K., and Sharf, A. A search-classify approach for cluttered indoor scene understanding. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* (2012).

[131] Nowozin, S., Gehler, P. V., and Lampert, C. H. On parameter learning in crf-based approaches to object class image segmentation. In *Proceedings of European Conference on Computer Vision (ECCV)* (2010).

[132] Ok, D., Kozinski, M., Marlet, R., and Paragios, N. High-level bottom-up cues for top-down parsing of facade images. In *IEEE International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)* (2012), pp. 128–135.

[133] Oliva, A., and Torralba, A. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision (IJCV) 42* (May 2001), 145–175.

[134] Oliva, A., and Torralba, A. Building the gist of a scene: the role of global image features in recognition. *Progress in brain research 155* (2006), 23–36.

[135] Parish, Y. I. H., and Müller, P. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 301–308.

[136] Payne, A., and Singh, S. Indoor vs. outdoor scene classification in digital photographs. *Pattern Recognition (PR) 38*, 10 (2005), 1533–1545.

[137] Prusinkiewicz, P. Graphical applications of l-systems. In *Proceedings on Graphics Interface '86/Vision Interface '86* (Toronto, Ont., Canada, Canada, 1986), Canadian Information Processing Society, pp. 247–253.

[138] Prusinkiewicz, P., and Lindenmayer, A. *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[139] Pullum, G., and Gazdar, G. Natural languages and context-free languages. *Linguistics and Philosophy 4*, 4 (1982), 471–504.

[140] Quack, T., Leibe, B., and Van Gool, L. World-scale mining of objects and events from community photo collections. In *CIVR* (2008), CIVR '08, ACM, pp. 47–56.

[141] Recky, M., Wendel, A., and Leberl, F. Façade segmentation in a multi-view scenario. In *IEEE International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)* (2011), pp. 358–365.

[142] Ren, X., and Malik, J. Learning a classification model for segmentation. In *ICCV* (2003), vol. 1, pp. 10–17.

[143] Riemenschneider, H., Bodis-Szomoru, A., Weissenberg, J., and Van Gool, L. Learning where to classify in multi-view semantic segmentation. In *ECCV* (2014).

[144] Riemenschneider, H., Krispel, U., Thaller, W., Donoser, M., Havemann, S., Fellner, D. W., and Bischof, H. Irregular lattices for complex shape grammar facade parsing. In *CVPR* (2012).

[145] Ripperda, N., and Brenner, C. Reconstruction of façade structures using a formal grammar and rjmcmc. In *Pattern Recognition - DAGM Symposium* (2006), pp. 750–759.

[146] Romer, C., and Plumer, L. Identifying architectural style in 3d city models with support vector machines. *Photogrammetrie - Fernerkundung - Geoinformation 2010* (2010), 371–384(14).

[147] Russell, S. J., Norvig, P., Candy, J. F., Malik, J. M., and Edwards, D. D. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[148] Salti, S., Petrelli, A., and Tombari, F. On the affinity between 3d detectors and descriptors. In *IEEE International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)* (2012).

[149] Sattler, T., Leibe, B., and Kobbelt, L. Fast image-based localization using direct 2d-to-3d matching. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)* (2011).

[150] Sengupta, S., Valentin, J., Warrell, J., Shahrokni, A., and Torr, P. Mesh Based Semantic Modelling for Indoor and Outdoor Scenes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2013).

[151] Shao, T. S. H., and Van Gool, L. Zubud-zurich buildings database for image based recognition, Technique report No. 260. Tech. rep., Swiss Federal Institute of Technology, 2003.

[152] Shechtman, E., and Irani, M. Matching local self-similarities across images and videos. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2007).

[153] Shechtman, E., and Irani, M. Matching local self-similarities across images and videos. In *CVPR* (2007).

[154] Shen, C.-H., Huang, S.-S., Fu, H., and Hu, S.-M. Adaptive partitioning of urban facades. *ACMGraphics 30*, 6 (2011), 184.

[155] Shi, J., and Malik, J. Normalized cuts and image segmentation. *PAMI 22*, 8 (Aug 2000), 888–905.

[156] Shieber, S. Evidence against the context-freeness of natural language. *Linguistics and Philosophy 8*, 3 (1985), 333–343.

[157] Shotton, J., Johnson, M., and Cipolla, R. Semantic texton forests for image categorization and segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2008).

[158] Shotton, J., Winn, J., Rother, C., and Criminisi, A. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *IJCV 81*, 1 (Jan. 2009), 2–23.

[159] Siagian, C., and Itti, L. Rapid biologically-inspired scene classification using features shared with visual attention. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 29* (February 2007), 300–312.

[160] Siagian, C., and Itti, L. Comparison of gist models in rapid scene categorization tasks. In *Proceedings of Vision Science Society Annual Meeting (VSS)* (May 2008).

[161] Simon, L., Teboul, O., Koutsourakis, P., and Paragios, N. Random exploration of the procedural space for single-view 3d modeling of buildings. *IJCV 93*, 2 (June 2011), 253–271.

[162] Simon, L., Teboul, O., Koutsourakis, P., Van Gool, L., and Paragios, N. Parameter-free/pareto-driven procedural 3d reconstruction of buildings from ground-level sequences. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012), pp. 518–525.

[163] Smith, T., and Waterman, M. Identification of common molecular subsequences. *Journal of Molecular Biology 147*, 1 (1981), 195–197.

[164] Socher, R., Lin, C. C., Ng, A. Y., and Manning, C. D. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *ICML* (2011).

[165] Song, S., and J., X. Sliding shapes for 3d object detection in depth images. In *ECCV* (2014).

[166] Stava, O., Benes, B., Mech, R., Aliaga, D. G., and Kristof, P. Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum 29*, 2 (2010), 665–674.

[167] Stiny, G. Pictorial and formal aspects of shape and shape grammars, 1975. Birkhauser Verlag, Basel.

[168] Stiny, G. Introduction to shape and shape grammars. *Environment and Planning B 7*, 3 (1980), 343–351.

[169] Stiny, G. Spatial relations and grammars. *Environment and Planning B: Planning and Design 9*, 1 (1982), 113–114.

[170] Stolcke, A. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, UC Berkeley, 1994.

[171] Summerson, J. *The Classical Language of Architecture*, 1 ed. MIT Press, 1996.

[172] Sunkel, M., Jansen, S., Wand, M., and Seidel, H.-P. A Correlated Parts Model for Object Detection in Large 3D Scans. *Proceedings of Eurographics (EUROGRAPHICS)* (2013).

[173] Szummer, M., Kohli, P., and Hoiem, D. Learning crfs using graph cuts. In *ECCV* (2008), pp. 582–595.

[174] Szummer, M., and Picard, R. W. Indoor-outdoor image classification. In *IEEE International Workshop on Content-Based Access of Image and Video Databases (CAIVD)* (Bombay, India, January 1998), p. 42–51.

[175] Talton, J. O., Lou, Y., Lesser, S., Duke, J., Měch, R., and Koltun, V. Metropolis procedural modeling. *Proceedings of the International Conference on Computer graphics and interactive techniques (SIGGRAPH) 30*, 2 (2011).

[176] Talton, J. O., Yang, L., Kumar, R., Lim, M., Goodman, N. D., and Měch, R. Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (2012).

[177] Taskar, B., Chatalbashev, V., Koller, D., and Guestrin, C. Learning structured prediction models: a large margin approach. In *ICML* (New York, NY, USA, 2005), ICML '05, ACM, pp. 896–903.

[178] Teboul, O. Ecole centrale paris facades database. http://vision.mas.ecp.fr/Personnel/teboul/data.php, 2010.

[179] Teboul, O. *Shape Grammar Parsing: Application to Image-based Modeling*. PhD thesis, Ecole Centrale Paris, 2011.

[180] Teboul, O., Kokkinos, I., Simon, L., Koutsourakis, P., and Paragios, N. Shape grammar parsing via reinforcement learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2011), pp. 2273–2280.

[181] Teboul, O., Kokkinos, I., Simon, L., Koutsourakis, P., and Paragios, N. Parsing facades with shape grammars and reinforcement learning. *PAMI 35*, 7 (2013), 1744–1756.

[182] Teboul, O., Simon, L., Koutsourakis, P., and Paragios, N. Segmentation of building facades using procedural shape priors. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010), pp. 3105–3112.

[183] Thomas, A., Ferrari, V., Leibe, B., Tuytelaars, T., and Van Gool, L. Depth-from-recognition: Inferring meta-data by cognitive feedback. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)* (2007).

[184] Tighe, J., and Lazebnik, S. Finding things: Image parsing with regions and per-exemplar detectors. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2013).

[185] Tighe, J., and Lazebnik, S. Superparsing - scalable nonparametric image parsing with superpixels. *International Journal of Computer Vision (IJCV) 101*, 2 (2013), 329–349.

[186] Tomita, M. Parsing 2-dimensional language. In *Current Issues in Parsing Technology*, M. Tomita, Ed., vol. 126 of *The Springer International Series in Engineering and Computer Science*. Springer US, 1991, pp. 277–289.

[187] Torralba, A. LabelMeDataset. http://people.csail.mit.edu/torralba/code/spatialenvelope/, 2010.

[188] Torralba, A., Murphy, K. P., Freeman, W. T., and Rubin, M. A. Context-based vision system for place and object recognition. In *ICCV* (Washington, DC, USA, 2003), ICCV '03, IEEE Computer Society, pp. 273–.

[189] Toshev, A., Mordohai, P., and Taskar, B. Detecting and parsing architecture at city scale from range data. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010), pp. 398–405.

[190] Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. Large margin methods for structured and interdependent output variables. *The Journal of Machine Learning Research (JMLR) 6* (Dec. 2005), 1453–1484.

[191] TUFTS University. Perseus digital library. http://www.perseus.tufts.edu/hopper/, 2010.

[192] Tyleček, R., and Šára, R. Spatial pattern templates for recognition of objects with regular structure. In *Pattern Recognition (PR)*, J. Weickert, M. Hein, and B. Schiele, Eds., vol. 8142 of *Lecture Notes in Computer Science*. Springer, Berlin Heidelberg, Germany, September 2013, pp. 364–374.

[193] Van den Bergh, M., Boix, X., Roig, G., de Capitani, B., and Van Gool, L. Seeds: superpixels extracted via energy-driven sampling. In *Proceedings of European Conference on Computer Vision (ECCV)* (2012).

[194] van Kaick, O., Xu, K., Zhang, H., Wang, Y., Sun, S., Shamir, A., and Cohen-Or, D. Co-hierarchical analysis of shape structures. *ACM Transactions on Graphics 32*, 4 (2013).

[195] Vanegas, C., Aliaga, D., and Benes, B. Building reconstruction using manhattan-world grammars. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010), pp. 358–365.

[196] Vergauwen, M., and Van Gool, L. Web-based 3d reconstruction service. *Machine Vision and Applications (MVA) 17*, 6 (2006), 411–426.

[197] von Luxburg, U. A tutorial on spectral clustering. *CoRR abs/0711.0189* (2007).

[198] Weissenberg, J., Riemenschneider, H., Prasad, M., and Van Gool, L. Is there a procedural logic to architecture? In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2013).

[199] Wendel, A., Donoser, M., and Bischof, H. Unsupervised facade segmentation using repetitive patterns. In *Proceedings of German Annual Symposium of Pattern Recognition (DAGM)* (2010), pp. 51–60.

[200] Wild, S. An earley-style parser for solving the rna-rna interaction problem (bachelor thesis), 2010.

[201] Wojek, C., and Schiele, B. A dynamic conditional random field model for joint labeling of object and scene classes. In *Proceedings of European Conference on Computer Vision (ECCV)* (2008), pp. 733–747.

[202] Wonka, P., Wimmer, M., Sillion, F. X., and Ribarsky, W. Instant architecture. *SIGGRAPH 22*, 3 (2003), 669–677.

[203] Wu, C. VisualSFM: A Visual Structure from Motion System. http://ccwu.me/vsfm/, 2011.

[204] Wu, C. Towards linear-time incremental structure from motion. In *International Conference on 3D Vision (3DV)* (2013).

[205] Wu, F., Yan, D.-M., Dong, W., Zhang, X., and Wonka, P. Inverse procedural modeling of facade layouts. *ACMGraphics 33*, 4 (July 2014), 121:1–121:10.

[206] Xiao, J., Fang, T., Tan, P., Zhao, P., Ofek, E., and Quan, L. Image-based façade modeling. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* (2008).

[207] Xiao, J., Fang, T., Zhao, P., Lhuillier, M., and Quan, L. Image-based street-side city modeling. *SIGGRAPH 28*, 5 (2009).

[208] Yang, C., Han, T., Quan, L., and Tai, C.-L. Parsing façade with rank-one approximation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012).

[209] Yang, M., and Förstner, W. A hierarchical conditional random field model for labeling and classifying images of man-made scenes. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on* (2011), pp. 196–203.

[210] Yang, M. Y., and Förstner, W. Regionwise classification of building facade images. In *Proceedings of the International Conference on Photogrammetric Image Analysis (PIA)* (2011), LNCS 6952, Springer, pp. 209 – 220.

[211] Younger, D. H. Recognition and parsing of context-free languages in time n3. *Information and Control 10*, 2 (1967).

[212] Zhang, C., Wang, L., and Yang, R. Semantic segmentation of urban scenes using dense depth maps. In *Proceedings of European Conference on Computer Vision (ECCV)* (2010).

[213] Zhang, H., Xu, K., Jiang, W., Lin, J., Cohen-Or, D., and Chen, B. Layered analysis of irregular facades via symmetry maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 32*, 4 (2013), 121:1–13.

[214] Zhang, Z. Microsoft kinect sensor and its effect. *IEEE MultiMedia 19*, 2 (Apr. 2012), 4–10.

[215] Zhang, Z., Zhang, Z., Robotique, P., and Robotvis, P. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and Vision Computing (IVC) 15* (1997), 59–76.

[216] Zhao, P., Fang, T., Xiao, J., Zhang, H., Zhao, Q., and Quan, L. Rectilinear parsing of architecture in urban environment. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010), pp. 342–349.

[217] Álvaro, F., Sánchez, J.-A., and Benedí, J.-M. Recognition of printed mathematical expressions using two-dimensional stochastic context-free grammars. In *International Conference on Document Analysis and Recognition (ICDAR)* (2011).

Curriculum Vitae

Anđelo Martinović was born in Zadar, Croatia on July 9, 1987. He obtained the degree of Master of Science in Computing, with highest honors, from the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia in July 2010. During his studies, he received the award for excellence in study two times, and the bronze plaque 'Josip Lončar' for the best students in the class. In September 2010 he joined the VISICS research group at the Department of Electrical Engineering, KU Leuven, Belgium as a predoctoral student. He started his PhD program in September 2011 under the supervision of prof. dr. Luc Van Gool. During his doctoral studies, he was involved in FP7 projects dealing with urban reconstruction and cultural heritage. He also assumed a teaching assistant role for the course 'Digital Electronics and Processors'. He has contributed several research papers in high-tier international conferences such as ECCV and CVPR.

List of Publications

Journal Articles

- M. Mathias*, **A. Martinović***, J. Weissenberg, and L. Van Gool. "ATLAS: A Three-Layered Approach to Facade Parsing," minor revisions under review at *International Journal of Computer Vision (IJCV)*

Peer-Reviewed International Conference Articles

- **A. Martinović***, J. Knopp*, H. Riemenschneider, and L. Van Gool. "3D All The Way: Semantic Segmentation of Urban Scenes From Start to End in 3D," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- **A. Martinović** and L. Van Gool. "Hierarchical Co-Segmentation of Building Facades," in *International Conference on 3D Vision (3DV)*, 2014.
- L. Van Gool, **A. Martinović** and M. Mathias. "Towards Semantic City Models," in *Proceedings of the 54th Photogrammetric Week*, 2013.
- **A. Martinović** and L. Van Gool. "Bayesian Grammar Learning for Inverse Procedural Modeling," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.
- **A. Martinović**, M. Mathias, J. Weissenberg, and L. Van Gool. "A Three-Layered Approach to Facade Parsing," in *European Conference on Computer Vision (ECCV)*, 2012. (Oral presentation)
- M. Mathias, **A. Martinović**, J. Weissenberg, S. Haegler, and L. Van Gool. "Automatic architectural style recognition," in *Proceedings of*

the 4th ISPRS International Workshop 3D-ARCH 2011: "3D Virtual Reconstruction and Visualization of Complex Architectures", 2011.

- M. Mathias, **A. Martinović**, J. Weissenberg, and L. Van Gool. "Procedural 3d building reconstruction using shape grammars and detectors," in *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, 2011.

Technical Reports

- **A. Martinović** and L. Van Gool. "Earley Parsing for 2D Stochastic Context Free Grammars," *Technical Report KUL/ESAT/PSI/1301*, KU Leuven, 2013.

* indicates equal contribution.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
PSI - VISICS
Kasteelpark Arenberg 10, box 2441
B-3001 Heverlee, BELGIUM
andelo.martinovic@esat.kuleuven.be
http://www.esat.kuleuven.be/psi/visics/