



HAL
open science

Shape grammar parsing: application to image-based modeling

Olivier Teboul

► **To cite this version:**

Olivier Teboul. Shape grammar parsing: application to image-based modeling. Signal and Image processing. Ecole Centrale Paris, 2011. English. NNT : 2011ECAP0024 . tel-00628906

HAL Id: tel-00628906

<https://theses.hal.science/tel-00628906>

Submitted on 6 Oct 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Microsoft®

ECOLE CENTRALE PARIS

PHD THESIS

to obtain the title of

Doctor of Ecole Centrale Paris

Specialty : APPLIED MATHEMATICS

Defended by

Olivier TEBOUL

Shape Grammar Parsing: Application to Image-based Modeling

prepared at Ecole Centrale de Paris, MAS laboratory

defended on June 1, 2011

Jury :

<i>Chairman :</i>	Pr. Marc SCHOENAUER	-	Inria Saclay
<i>Reviewers :</i>	Pr. Jiri MATAS	-	Czech Technical University, Prague
	Pr. Marc POLLEFEYS	-	ETH Zurich
<i>Advisor :</i>	Pr. Nikos PARAGIOS	-	Ecole Centrale de Paris
<i>Examiners :</i>	Pr. Carsten ROTHER	-	Microsoft Research Cambridge
	Pr. Sylvain LEFEBVRE	-	Inria Nancy
	Pr. Renaud KERIVEN	-	Ecole Polytechnique - Acute3D
	Pr. Iasonas KOKKINOS	-	Ecole Centrale Paris

Acknowledgments

First of all, I would like to sincerely acknowledge my advisor Nikos Paragios, both as a supervisor and as a person, for his strong support, the total liberty he gave me and the numerous opportunities he offered me along the way. I had a wonderful time working in Ecole Centrale Paris under his supervision during these three years and I will not forget it. Then I would like to thank Microsoft France and Microsoft Research Cambridge, and more specifically Pierre-Louis Xech and Fabien Petitcolas, for having supported my work with a complete trust and freedom, putting me in the best possible conditions to carry on my research.

Second, I would like to thank all the committee members, the two reviewers Jiri Matas and Marc Pollefeys, the chairman Marc Schoenauer, and the examiners Carsten Rother, Renaud Keriven, Sylvain Lefebvre, and Iasonas Kokkinos for having taken the time to read and evaluate my work, for their relevant comments and questions and for the fruitful remarks they provided in their reports and during the defense.

Third, I would like to acknowledge the people whom I have collaborated the most with, that is to say Loïc Simon and Panagiotis Koutsourakis. We spent a lot of time together, shared a lot together and therefore I owe them a lot both professionally and personally. Moreover, I had a great pleasure working with Iasonas Kokkinos on very exciting problems. Iasonas has really been a second adviser to me and he has all my gratitude. I would also like to thank my colleagues that bore me in the same room during these years with great patience and kindness and I really consider them as true friends: Aris Sotiras, Wang Chaohui, Loïc Simon, Panos Koutsourakis, Ahmed Besbes, Salma Essafi and Radhouene Neji. My gratitude also goes to my colleagues who reviewed my work: Ahmed, Aris, Chaohui, Loic and Sarah Parisot, and on a larger scale, I would like to thank all the members of the vision and medical imaging team: Maxime, Noura, Martin, Charlotte, Olivier, Regis, Daniel, Mickael, Georg, Fabrice, Nicolas, Pierre-Yves, Katerina, Bo, Sarah, Stravros and Haithem with whom it was a pleasure to work on a daily basis. I do not forget people who visited the lab for a while but long enough to become friends: Kostas who supervised my work at the beginning and with whom it is always a pleasure to share a coffee whenever I visit Athens, Rola from Montreal and José from Barcelona.

Then, I would like to acknowledge all the members of the MAS laboratory, with special thanks to Sylvie Dervin who has always been one of my first support, Erick Herbin, Marc Aiguier, Céline Hudelot and Florian De Vuyst who offered me the possibility to teach at Ecole Centrale Paris, Paul-Henry Cournède, Frederic Abergel, Patrick Callet and Gilles Faye for their kindness and with whom it is always a pleasure to discuss.

Moreover, I would like to thank the people from outside whom I had the great opportunity to collaborate with. First of all Sylvain Lefebvre with whom I really appreciated to collaborate on a very exciting project and that invited me along with Georges Drettakis to their lab in Sophia Antipolis in 2009. Then Giorgos Tziritas and Nikos Komodakis who invited me in Heraklio, Crete during the summer 2009 where I had a great time on a personal and professional level. Alan Yuille who invited me in UCLA in summer 2010 and with whom I had a great pleasure to take the time to discuss deeply about vision problems.

Outside of the professional sphere, I would like to thank all my family who always showed me love and support and who were most important to me that I may have showed. In particular, I would like to thank Aga who has always been my first and unconditional support, always bringing me love and happiness. Beyond that, you all helped me in your own way and to all of you I dedicate this thesis.

There are also some friends that I consider as family and that I would also like to acknowledge. They all supported and helped me, sometimes without even knowing it. Ali Ezzatyar, Razvan Ionasec, Bastien Grandcoin, Miloud Chahlaoui, Remy Beharaing, Sinan Al Awabdh, Alba Jimenez and Christina Papista are all like brothers and sisters and they have my everlasting gratitude.

Ultimately, I would like to thank all my friends and the people that shared some time with me during my Ph.D, making me forget about my research for a while. Those are my numerous friends from Paris wherever they initially come from, my Volleyball team, people from Crete that hosted me in their beautiful island, all the very nice people I met in California, my Brazilian friends and my Portuguese teacher. Last but not least, I would like to thank Luiza Machado for all she brought me and who certainly changed my life more than anybody else during these three years.

Abstract

The purpose of this thesis was to perform facade image parsing with shape grammars in order to tackle single-view image-based 3D building modeling. The scope of the thesis was lying at the border of Computer Graphics and Computer Vision, both in terms of methods and applications.

Two different and complementary approaches have been proposed: a bottom-up parsing algorithm that aimed at grouping similar regions of a facade image so as to retrieve the underlying layout, and a top-down parsing algorithm based on a very powerful framework: Reinforcement Learning. This novel parsing algorithm uses pixel-wise image supports based on supervised learning in a global optimization of a Markov Decision Process.

Both methods were evaluated quantitatively and qualitatively. The second one was proved to support various architectures, several shape grammars and image supports, and showed robustness to challenging viewing conditions; illumination and large occlusions. The second method outperformed the state-of-the-art both in terms of segmentation and speed performances. It also provides a much more flexible framework, in which many extensions may be envisioned.

The conclusion of this work was that the problem of single-view image-based 3D building modeling could be solved elegantly by using shape grammar as a Rosetta stone to decipher the language of Architecture through a well-suited Reinforcement Learning formulation. This solution was a potential answer to large-scale reconstruction of urban environments from images, but also suggested the possibility of introducing Reinforcement Learning in other vision tasks such as generic image parsing, where it have been barely explored so far.

Keywords: Computer Vision, Computer Graphics, Procedural Modeling, Image Parsing, Reinforcement Learning, Image-based Modeling.

Résumé

L'objectif de cette thèse était de résoudre le problème d'analyse d'image de façade avec a priori de forme procédurale en vue de l'appliquer à la modélisation 3D d'immeuble à partir d'une seule image. Le cadre de cette thèse se situe à la frontière de l'informatique graphique et de la vision par ordinateur, tant d'un point de vue des méthodes employées que des applications potentielles.

Deux approches complémentaires ont été proposées: une méthode dite ascendante qui cherche à regrouper des régions similaires de l'image afin de révéler la structure sous-jacente de la façade ; et une méthode dite descendante basée sur les puissants principes de l'apprentissage par renforcement. Ce nouvel algorithme combine des mesures locales issues de méthodes d'apprentissage supervisé dans une optimisation globale d'un Processus de Décision Markovien, qui découvre la grammaire du bâtiment au fil des itérations.

Ces deux méthodes ont été évaluées qualitativement et quantitativement. Les résultats ainsi obtenus, se sont avérés bien meilleurs que l'état de l'art sur le plan de la rapidité, de la qualité de segmentation, mais également au niveau de la flexibilité de la méthode et de ses extensions éventuelles. Cet algorithme a été abondamment testé sur différents types de grammaires de formes, sur différents styles architecturaux, avec différentes mesures sur les images, et s'est avéré particulièrement robuste aux conditions d'illuminations et aux occlusions.

En conclusion, les grammaires de formes peuvent être utilisées comme une pierre de Rosette afin de déchiffrer le langage de l'architecture et permettent ainsi de modéliser un bâtiment 3D à partir d'une unique image, à travers un nouvel algorithme issu de l'apprentissage par renforcement. D'une part la méthode développée apporte une réponse au problème de reconstruction urbaine 3D à large échelle à partir d'images, et d'autre part elle laisse entrevoir de potentielles applications de l'apprentissage par renforcement en vision par ordinateur, domaine qui jusqu'alors ne s'y était que très peu intéressé.

Mots clés: Vision par ordinateur, informatique graphique, modélisation procédurale, analyse d'images, apprentissage par renforcement, modélisation à partir d'images.

Contents

1	Introduction	19
1.1	Industrial Context: The 3D Industry	19
1.1.1	A Fast Growing Industry	19
1.1.2	3D Content Creation and its Challenges	20
1.2	State-of-the-art in Image-based 3D Modeling	21
1.2.1	Fully Automatic Multiple-Views Methods	21
1.2.2	Semi-automatic Methods	24
1.3	Theoretical Context: the Semantic Gap	25
1.4	Objectives	26
2	Procedural Modeling	27
2.1	State of the Art	27
2.1.1	Chomsky’s Formal Grammars	28
2.1.2	L-Systems	30
2.1.3	Shape Grammars	32
2.1.4	Set Grammars in Computer Graphics	33
2.2	Procedural Shapes	35
2.2.1	Definitions	36
2.2.2	Derivation process	38
2.2.3	Operators	38
2.2.4	Simple Examples	44
2.3	Building Modeling with Shape Grammars	46
2.3.1	Roof Operators	46
2.3.2	Tag Operators for Consistent Shape Generation	49
2.3.3	Examples of Shape Grammars	51
2.4	2D Binary Split Grammars	53
2.4.1	Definitions	53
2.4.2	Expressive Power of BSGs	55
2.4.3	BSG for Compact Accurate Modeling	55
2.4.4	Some Examples of BSGs	56

2.5	Conclusion	58
3	Bottom-Up Parsing	63
3.1	State of the Art	63
3.1.1	Introduction	63
3.1.2	Repetitions and Lattices	64
3.1.3	Axial Symmetries	65
3.1.4	Window Detection	66
3.1.5	Relaxing the Regularity Constraint	66
3.2	Detection of Similar Features	67
3.2.1	Interest Point Detection	67
3.2.2	Interest Point Description	68
3.2.3	Clustering in the Features Space	70
3.2.4	Descriptor Clustering	72
3.3	Cluster Analysis	73
3.3.1	Position Distribution	74
3.3.2	Jump Distribution	74
3.3.3	Cluster Score	75
3.3.4	Global Position and Jump Distributions	76
3.4	Modeling Pattern Repetitions	76
3.4.1	Markov Chain Formulation	77
3.4.2	Unary Potentials	78
3.4.3	Pair-wise Potentials	78
3.4.4	Optimization	79
3.5	Experimental Validation	79
3.5.1	Qualitative Results	80
3.5.2	Quantitative Validation	80
3.5.3	Failure Cases	82
3.6	Conclusion	82
4	Top-Down Parsing: A Novel Algorithm	85
4.1	State of the Art	85
4.1.1	Context-Free Grammar Parsing	86
4.1.2	Image Grammar Parsing	86
4.1.3	Facade Parsing	87
4.2	A Brief Overview Markov Decision Processes	89
4.2.1	Markov Decision Processes	90
4.2.2	Dynamic Programming	94
4.2.3	Monte-Carlo	97
4.2.4	Reinforcement Learning	100
4.2.5	Semi Markov Decision Processes	103

4.3	Parsing Split Grammars	104
4.3.1	The Parsing Problem	104
4.3.2	1D parsing	105
4.3.3	2D Parsing	110
4.4	Going Further with Reinforcement Learning	116
4.4.1	Data-Driven Policy	117
4.4.2	Approximate Reinforcement Learning	119
4.4.3	User-Defined Constraints	124
4.4.4	Minimum Description Length	124
4.4.5	Grammar Selection	127
4.5	Conclusion	127
5	Top-Down Parsing: Applications	129
5.1	Defining Reward Functions on Real Images	129
5.1.1	Perfect Rewards for Artificial Data	129
5.1.2	Randomized Forest for Consistent Architectural Styles	130
5.1.3	Gaussian Mixture Models	132
5.1.4	Unsupervised Hue based Rewards for Binary Parsing	133
5.1.5	Depth Maps	135
5.2	Quantitative Validation	137
5.2.1	Convergence on Real Data	137
5.2.2	A Measure for Segmentation	139
5.2.3	A Measure for Topology	139
5.2.4	Paris Benchmark 2010	140
5.2.5	Paris Benchmark 2011	143
5.2.6	Robustness to Noise	144
5.3	Qualitative Validation	146
5.3.1	Binary Segmentation	146
5.3.2	4-colors Facades	147
5.3.3	Hausmannian Architecture	148
5.3.4	Modern Architectures	149
5.3.5	Robustness to Occlusions	149
5.3.6	Robustness to Illumination	150
5.3.7	From 2D to 3D: Image-based Procedural Building Modeling	151
5.4	Conclusion	153
6	Conclusion	155
6.1	Contributions	155
6.2	Applications	156
6.3	Future Work	157

List of Figures

2.1	Example of context-free grammar	30
2.2	A L-system for Von Koch's curve.	31
2.3	A shape grammar for greek cross church plans	33
2.4	Illustration of the scope transformation	37
2.5	3D Transformation operators	40
2.6	Split Operator	41
2.7	Repeat and Mirror splits	42
2.8	Stairs grammar example	43
2.9	Cantor's dust and Menger's sponge shape grammars.	44
2.10	Generation of a tree using a shape grammar.	45
2.11	Straight skeleton algorithm	47
2.12	Roofs operators	48
2.13	Consistency Tags	49
2.14	Procedural Greek Doric Temples	51
2.15	Step-by-step construction of an Haussmannian Buildings.	52
2.16	Procedural Haussmannian Buildings	53
2.17	A shape generated by a BSG and its corresponding parse tree	54
2.18	Modeling a floor using mutually recursive binary splits.	56
2.19	3 BSGs	57
3.1	Different interest point detectors	68
3.2	Sift descriptors	70
3.3	Clustering the SIFT descriptors	73
3.4	Position distribution in a cluster	74
3.5	Jump distribution in a cluster	75
3.6	Complete distributions	77
3.7	The Markov Chain formulation of the pattern repetition.	78
3.8	Some Results of bottom-up parsing	79
3.9	Examples of robustness to occlusions.	80
3.10	Some results on European facades	81

3.11	Examples of building from different American cities.	82
3.12	Failure Cases	83
4.1	Agent-environment interaction scheme.	90
4.2	ϵ -greedy and softmax policies with respect to Q	99
4.3	Example of image to be parsed with a 1D split grammar.	105
4.4	The parse tree as a decision process	108
4.5	Rewards function in the 1D case	109
4.6	Comparison of DP and RL on the 1D parsing	110
4.7	A complete derivation in DFS order	111
4.8	State Aggregation	112
4.9	Convergence of the RL parsing algorithm	116
4.10	Representation of the h_X and h_Y functions	117
4.11	Speed-up with data-driven exploration..	118
4.12	Example of Radial Basis Function Networks (RBFN).	122
4.13	Learning Curve with Function Approximation.	123
4.14	Different parse tree obtained for different MDL penalty	126
5.1	A Randomized Tree	131
5.2	Merit functions for the Hausmann BSG.	132
5.3	Merit functions using GMM	133
5.4	Analyzing a facade through the Hue	135
5.5	Creating a Depth Map from a 3D Points Cloud.	136
5.6	Convergence of the algorithm on real data.	138
5.7	Comparing the segmentations obtained by 3 different methods.	141
5.8	Robustness to salt-and-pepper noise.	145
5.9	Parsing images with binary segmentation grammar based on Hue rewards.	146
5.10	Parsing images from New York City, USA.	147
5.11	Barcelona and Budapest Data Sets parsed using the 4-colors grammar.	148
5.12	Higher Buildings.	149
5.13	Parsing Hausmannian buildings with a Hausmannian BSG	150
5.14	Robustness to artificial occlusions	151
5.15	Robustness to natural occlusions	151
5.16	Robustness to cast shadows	152
5.17	Robustness to night illumination	152
5.18	From 2D parsing to 3D models	154

List of Tables

2.1	The binary floor BSG	55
2.2	A binary grammar: wall/window.	57
2.3	The 4-colors grammar: wall/window/roof/shop.	58
2.4	Doric Temple Shape Grammar.	60
2.5	Hausman Buildings Shape Grammar.	61
3.1	Dice Coefficients and topology precision averaged on the NYC dataset.	81
4.1	Step-by-step episode of the 1D parsing	106
4.2	A repeat grammar for parsing 1D floors	125
4.3	Result of parsing with MDL penalty	126
5.1	Parsing algorithm performances on the full Monge dataset	143

List of Algorithms

2.1	Derivation Process	38
2.2	Consistent Derivation Process	50
4.1	Value Iteration for a Deterministic MDP	94
4.2	Monte-Carlo method for solving MDP	98
4.3	SARSA	101
4.4	Q-Learning	102
4.5	Parsing BSG via Q-Learning	115
4.6	Stochastic Gradient Descent Q-Learning	121

Chapter 1

Introduction

The present work lies at the border of two worlds: Computer Graphics and Computer Vision. It aims at understanding images in order to create 3D models of buildings from single views using shape grammars as a Rosetta stone for building images. At first glance, this problem seems to be by nature geometric: bridging the gap between 2D and 3D. A single view is theoretically not enough to retrieve the depth of the model. However, the prior knowledge of observing a building and the use of a shape grammar to describe it, turns the geometric problem into a semantic one. This problem is at the core of Computer Vision since the very beginning: the semantic gap.

In this introduction, we are going first to introduce the industrial context of my work in section 1.1, and review the classical methods brought so far by the Computer Vision community to bridge the gap between 2D and 3D in section 1.2. Then, we will explain in section 1.3 why the problem is not only interesting in terms of industrial application, but also on a more theoretical level. The Holy Grail of Computer Vision remains the problem of giving sense to a raw matrix of pixels the way our brain turns the optical signals sensed by the eye into meaningful and sensible information and ideas. Eventually, we will formulate the objectives of the thesis in section 1.4 and give an overview of the work done in order to reach them.

1.1 Industrial Context: The 3D Industry

1.1.1 A Fast Growing Industry

Historically, the 3D industry started to grow amid the cinema industry in the 1970's. *Star Wars* for instance was one of the first movies to use some 3D animation. The same decade saw the birth of SIGGRAPH which still remains the first conference of Computer Graphics. In the 1980's, the cinema kept on using more and more 3D technologies, but the 3D industry really took off in the 1990's. The first totally computer animated movies became block blusters, video games went from 2D to 3D and graphics software became more and more popular. This very fast increase of 3D technologies is directly linked to the increasing power of CPUs and GPUs together with the

decreasing cost of the hardware. In the 1980's a few companies could afford the machines to run 3D design software, whereas ten years later everybody can manipulate them.

In the 2000's, the 3D industry is expanding fast and reaches more and more fields of activity. The Internet now rules the market and helps 3D applications to spread out fast. Google maps, Microsoft Bing Maps are integrating 3D models and 3D contents to their virtual worlds. Moreover, the Internet is now in everybody's pocket, and smart phones are now small computers. 3D applications have to run on embedded devices: cell phone, navigation systems or video games portable consoles. Very recently, 3D is getting even more real with new devices that create an illusion of depth based on stereoscopic principle: 3D movie theaters and 3D TV screens are more and more popular. What was a gadget yesterday could be part of our daily life tomorrow.

In 20 years, the 3D industry went from a very exotic niche, gathering few companies, few employees and few customers, to a global industry, employing thousands of people world-wide. According to IDATE, the video games market represents in 2010 almost 50 billions of dollars, around 10 times more than a decade ago. The video game industry has overtaken the cinema industry and some video games have bigger budgets than Hollywood blockbusters. With even more applications and needs for 3D coming with the release on the market of 3D guzzling devices, no need to say that the 3D industry has some good days ahead.

1.1.2 3D Content Creation and its Challenges

Since 3D applications are mushrooming, the need for 3D content is growing as well. 3D applications need 3D content, and this may be one of the bottlenecks of the 3D industry expansion. The new challenges of 3D content creation are:

Easy content creation So far, manipulating and designing 3D models is a time consuming process. Students are formed on some software, and are hired on the job market for their expertise in very specific, tedious and technical tasks.

Large scale production Knowing that modeling a single highly detailed 3D model takes time, large scale modeling needs methods to decrease the modeling time, without deteriorating the model quality.

Realistic modeling How to obtain 3D models that are recognisable? How to use real data and real measurement to guide the modeling process?

Low cost How to fulfill these requirements at low cost? How to replace a team of modeling experts by automated tools?

Nowadays, graphic artists are creating 3D contents based on three different design paradigms:

Computer Assisted Design (CAD) uses software such as Maya, AutoCAD, Blender or 3D Studio Max that require a real deep expertise.

Sketching is relatively more recent and gets rid of the necessary specialized knowledge of classic modeling. It aims at creating fast approximate 3D models with a small amount of very intuitive interaction. Google SketchUp for instance implements the ideas developed by [Zelevnik 2006].

Procedural Modeling describes 3D geometry with shape grammars (see chapter 2). Unlike sketching or CAD, they need a heavy human investment at first to design a shape grammar, but can then be applied to generate at low cost complex and structured geometries. In [Gips 1999], James Gips one of the two pioneers of shape grammars defines the three problems of shape grammars: automatic shape generation, shape grammar parsing and shape grammar inference. The first problem is to some extent solved, and it aims at generating shapes from formal shape grammar definitions. The second problem aims at explaining an input shape with a given shape grammar, and the last problem which is also the most challenging is to create shape grammars from a set of shapes.

None of these three paradigms take up all of the above challenges of 3D content creation. And it will probably be so, as long as the user will be intensively involved in the design loop. Thus, the main idea to tackle these new challenges is to process automatically images to create 3D contents. Images are now flooding our lives, whatever their natures: satellites keep taking pictures of our world, people are taking pictures of their lives and of the places they live in, and are sharing them on the Internet. Furthermore, some companies such as Google Street View are taking pictures of our streets. Although this huge amount of information is usually available for free, it is, so far, barely analyzed.

Many industries have decided to make use of the power of images and the cheap availability of image data to automate their processes. Besides, images are measurements of the reality, and I believe there is no better way to get realistic data than to create them based on real measurements.

1.2 State-of-the-art in Image-based 3D Modeling

In the literature, there exist several ways to tackle image-based 3D modeling or 3D reconstruction. Some are fully automatic and based on multiple views whereas others take advantage of user interactions to understand the nature of the 3D scene better. Let's review these two approaches.

1.2.1 Fully Automatic Multiple-Views Methods

Automatic image-based modeling is a well-known problem in Computer Vision. In a multi-view approach, given N images of the same scene, taken from different points of view, the goal is to retrieve the 3D structure of the scene as well as the relative positions of the cameras.

Perspective Reconstruction

Even if the equations were already formulated by Kruppa in 1913, the problem was completely studied and explained in the 1990's by the Computer Vision community, among others Richard Hartley [Hartley 1992], Olivier Faugeras [Faugeras 1993] and Andrew Zisserman [Hartley 2003]. However, the solution they propose is up to a perspective transformation. This perspective ambiguity limits the scope of the method.

Metric Reconstruction

In order to obtain realistic models that can be used in real world applications, the reconstruction needs to be Euclidean. In other words, all the solutions are equivalent under a uniform scale transform. In 1998, Marc Pollefeys introduces in [Pollefeys 1998] the first automatic metric reconstruction methods, by assuming that some parameters are known and others can vary in the self-calibration step. The method proposed by Pollefeys to obtain dense 3D models from uncalibrated cameras is the generic scheme that will be followed by most of the methods thereafter: (i) extract interest points in all the images, (ii) match the points, (iii) compute a sparse 3D points cloud from the correspondences, (iv) refine to a dense 3D points cloud. This work was also extended in [Pollefeys 2001, Pollefeys 2002].

From then on, many approaches were proposed in order to increase the quality of the reconstruction, to speed-up the process or to alleviate the need of taking many images of the same scene, taking advantage of the pictures available on the Internet. The increasing number of methods available called for a proper comparison tool. For that matter, in 2006 a complete benchmark containing image sequences, ground truths and evaluation tools was released, on which all the methods are now fairly competing [Seitz 2006].

Shape Priors

One way to improve the reconstruction quality is to introduce shape priors. The idea is to make some assumptions about the scene being reconstructed. Indeed, the noise due to the geometric and photogrammetric image measurements is by nature bound to introduce noise in the dense reconstruction. Therefore, if we expect the 3D scene to have some geometric properties, then this prior has to be introduced in the reconstruction framework in order to be enforced in the final model.

A first natural shape prior is to suppose the model piece-wise planar. Many works follow this assumption, such as [Baillard 1999, Dick 2000, Bauer 2002, Fraundorfer 2006, Sinha 2008, Sinha 2009]. In these papers, structure-from-motion is used as a starting point to retrieve 2D lines, then 3D lines and finally 3D planes. The optimization procedures in these works may vary. Some works formulate the piece-wise planar reconstruction as a MRF where each label corresponds to a different plan hypothesis. Piece-wise planar models are very often well-suited to describe man-made environment, but the assumption may sometimes be too strong for natural or hybrid scenes. For that matter, Ponce and Furukawa replace it by a reconstruction made of small rectangular

patches [Furukawa 2010], whereas [Hiep 2009] propose to use tetrahedral meshes to describe the volume of the 3D scene. Each tetrahedron can be filled or not, and the optimal binary labeling of the scene is optimized by graph-cut algorithm [Boykov 2001].

More recently, hybrid approaches have also been proposed, either combining piece-wise planar and non-planar parts [Gallup 2010], also introducing a necessary segmentation, or combining generic meshes with parameterized primitives [Lafarge 2010] calling for more complex optimization procedures.)

While trying to increase the quality of the 3D reconstructions, people have also worked on increasing the speed performances, especially because reconstruction from a large number of images turn out to be very time consuming, at each step of the scheme defined by [Pollefeys 1998].

Fast Reconstructions

Several improvements have been proposed to boost the computational time of 3D reconstructions from many images. The heavy step of feature points matching has been improved with approximate nearest neighbors by [Arya 1998] while [Lourakis 2004] use a sparse bundle adjustment to accurately retrieve a points cloud from a few correspondences.

Other bottle-necks have been removed by [Mordohai 2007, Frahm 2009]. In this work they use GPU implementations of 2D feature tracking and plane sweeping, and manage to reach real-time 3D reconstruction from video sequences. In [Cornelis 2006], videos are also processed to perform reconstruction. The difference with previous works is that they combine it with a car detector, so that the detection helps the reconstruction by detecting occlusions, and the reconstruction helps the detection by introducing some understanding of the 3D scene. This is a very good example of how two very complex problems are better solved when they are considered jointly.

Eventually, [Van Meerbergen 2002] proposed another way to speed up the computations in a coarse-to-fine approach to get faster correspondences on the epipolar lines using dynamic programming.

All these improvements are the direct consequences of the democratization of 3D metric reconstruction and its application to large databases. In this trend, some very interesting works on the use of the Internet, as an almost infinite database, have raised many problems and brought very elegant solutions and impressive large scale reconstructions of entire cities.

Reconstruction from Huge Databases

One of the most significant limitations of multi-view reconstruction from images is the acquisition of the images. However, nowadays, websites such as Flickr offer free on-line storing of photographs. Knowing that every tourist visiting Notre-Dame de Paris or the Parthenon takes dozens of photographs and that these monuments are visited by thousands of tourists every day, Snavely came up with the idea that building his own database was pointless, since he could as well help himself on the Internet [Snavely 2006]. The huge number of images available brings new challenges. Taking advantage on the advance in interest point descriptors [Brown 2005], feature matching [Arya 1998]

and bundle adjustment [Lourakis 2004], they build very large scale points clouds up to an entire city [Agarwal 2010] and propose a way to navigate between the images through a skeletal graph [Snavely 2008]. Related to this work, they developed Photosynth, which was recently integrated in Microsoft Bing Maps.

These reconstructions are based on different images, taken at different times of the year and of the day and by different people with different cameras. Despite these challenging conditions, they manage to build an efficient reconstruction of the scene and retrieve the positions of the cameras. However, the proposed reconstruction shows no structure it remains a points cloud that is perfect as a support for the image database, and provide an intuitive way to traverse it, but that is not sufficient as a 3D model.

Limitations of Multi-View Reconstruction

The idea of metric reconstruction is very appealing, and the quality of the reconstruction methods has been rocketing for the last decade, both in terms of 3D models and in terms of speed. However, the criticism I made about Snavely's work about the lack of structure is extensible to multi-view reconstruction in general. Most of the methods create soups: soups of 3D points, soups of polygons or soups of primitives. Even if in the last case some structure arises, the obtained models still lacks semantics. Indeed, this downside cannot be alleviated as long as no further assumptions are made about the 3D scene or as long as the reconstruction methods are not associated with some kind of scene or image parsing.

The lack of structure and semantics in the 3D models make them difficult to post-process manually. These representations are flat and the reconstruction of the background, or outliers are not distinguished from the reconstruction of the foreground objects. These methods catch the brute reality of the scene but not its core. For example, from a reconstruction of a building, they would not be able to construct another building that is inspired from the reconstructed one. They do not catch the way the geometry is organized, mainly because they lack semantic information.

Furthermore, in any industrial application, fully automatic methods are not that attractive, since once they fail they can hardly be modified by a human intervention. On top of that, fully automatic methods are computationally heavy, time-consuming and need many images to perform an accurate reconstruction. While dealing with man-made environment, it is somehow frustrating to observe that a human user would need less time and fewer images to perform more accurate reconstructions. Besides, such a man-made reconstruction would be also easier to modify, since the 3D models would carry semantic information. Therefore, semi-automatic methods made their paths through 3D reconstruction, by allowing the user to guide the algorithm.

1.2.2 Semi-automatic Methods

Since retrieving accurately the structure of a 3D scene in 2D images is sometimes very tricky while a user has no trouble identifying the objects in an image, semi-automatic methods naturally appeared as powerful reconstruction alternatives. One of the first works to do so has been done in

the mid 1990's [Debevec 1996]. In this method, the user draws some corresponding edges in the different images in order to guide the creation of the 3D model. This is used to create accurate user-assisted 3D models with view depending texture mapping for high quality rendering. [Oh 2001] proposes an editing system with a single image that enables to recover the depth of the objects and their illumination properties in order to synthesize new points of view. In [Hengel 2006], the authors edit video. With a few clicks in some frames, they manage to recover the geometry of the 3D scene, and use it for augmented reality applications. Eventually, in [Van Den Hengel 2007] the authors present an approach called VideoTrace to build 3D models from manually segmented objects in videos. As often in these works, the approach combines automatic and user-assisted tools that enable them to get very impressive results with a reasonable level of interaction.

We have formulated so far the need of the 3D industry for efficient methods in 3D content creation, and the current solutions to this problem. We have insisted on the need for semantic information in order to overcome the limitations of 3D reconstruction and image-based modeling methods. In Computer Vision, a vast amount of theoretical works deal with retrieving automatically this semantic information. The problem is called image parsing and constitutes one of the most challenging unsolved problems of Computer Vision.

1.3 Theoretical Context: the Semantic Gap

In Computer Vision, understanding the content of the images has always been a hot topic, not to say the main issue. A universal image parser would have an infinite number of industrial applications in many fields of human activity. Furthermore, it would also give us some clues about the way the brain parses the images it receives, and therefore would be another step in understanding the way our brain works.

Unfortunately, such an algorithm does not exist yet and even though computer scientists, mathematicians, cognitive scientists have taken part in Computer Vision, the problem is far from being solved. Indeed, this Holy Grail of Computer Vision combines almost all the different problems of vision in a single one: low-level, mid-level and high-level vision.

In this thesis, we are not going to tackle the full image parsing problem, but a simpler one: facade parsing using shape grammar. This may seem to be a little limited at first glance, but gathers many interesting theoretical problems: on the one hand, a building geometry is quite constrained, any geometry cannot be envisioned. On the other hand, the variability of building layouts and geometries is almost infinite. For that matter, the underlying image grammar can be formulated with a split grammar, providing a challenging but reasonable amount of complexity. We do believe that replacing the full image parsing problem by a simpler one is a very fruitful path. It enables us to understand and identify the intrinsic difficulties of the full problem on a tractable case, and to propose specific solutions that could later be applied to the generic image parsing problem. Therefore, from a theoretical point of view, solving the facade parsing problem constitutes another step towards bridging the semantic gap, and from an industrial point of view, it brings a concrete solution to a very hot problem of the 3D industry.

1.4 Objectives

The objectives of the present work are to take advantage of the world of Computer Graphics and the world of Computer Vision in order to bring a solution to problems of both worlds: single view image-based 3D building modeling, and facade parsing.

From the Computer Graphics perspective, the objective is to tackle the second problem defined by James Gips: shape grammar parsing. Given a single input image of a building, how to turn it into a 3D model provided a shape grammar. Such an algorithm would satisfy the four challenges: it would easily create 3D models at low cost, since it needs a single image and no user interaction; it would therefore be applicable to large scale modeling and would provide realistic models based on real data. Moreover, since the 3D models would be derived from a shape grammar, modifying or enhancing them would be very easy. These models genuinely carry semantic information that can be more easily handled by a human user than mere meshes.

From the Computer Vision perspective, solving shape grammar parsing would be another stone on the bridge over the semantic gap between raw image data to semantic interpretation of images.

To overcome these challenges, we will start by defining procedural modeling in chapter 2, and explain how entire classes of buildings can be caught by a single shape grammar. Then we will propose two different and complementary solutions to facade parsing. In chapter 3, we present a bottom-up approach of the problem. It aims at grouping automatically image regions so as to build a complete binary segmentation of the image. This algorithm shows indeed some severe limitations that prevent us from using it for accurate multi-class segmentation with complex shape grammars. Therefore, chapter 4 introduces a novel top-down algorithm that performs efficient parsing of split grammars, and chapter 5 explains how to apply it on real images and how to use this parsing algorithm in image-based procedural modeling of buildings. Ultimately, the conclusion closes this doctoral work by a discussion about the main contributions and their potential industrial and theoretical impacts, and presents several extensions that could be envisioned as future work.

Chapter 2

Procedural Modeling

This chapter presents a dynamic way of describing complex but structured geometries called *Procedural Modeling*. Some shapes can be very complex in a descriptive representation, but turn out to be much simpler in a dynamic one that incrementally builds the shape through basic operations. Fractals, plants or buildings are good examples of such geometries: they can be seen as the output of a recursive procedure that sequentially replaces some simple parts by other simple parts. During this process, the global complexity of the shape as well as the number of operations keep increasing, but each operation still involves only basic geometry. Like LEGO, the power of procedural modeling lies in the composition of simple rules that encode the semantic-geometric relationships, and not in the intrinsic complexity of the involved elements.

After reviewing the state-of-the-art of procedural modeling in section 2.1, section 2.2 explains how to build a procedural engine that is capable of randomly generating a whole language of shapes from a grammar. Section 2.3 presents specific operators for procedural building modeling and shows two examples of shape grammars for architectural 3D modeling: Doric temples and Haussmannian buildings. Eventually, we present a novel kind of 2D grammars in section 2.4 called Binary Split Grammars (BSGs) that are efficiently capable of modeling 2D facades through mutually recursive binary rules.

2.1 State of the Art

Geometry is one of the oldest science of all. Informally started in ancient Egypt and Babylon around 3000 B.C. and already for urban and architectural needs, it was then properly formalized in ancient Greece around 600 B.C. mainly by Thales, Pythagoras and their disciples. Some 300 years later, Euclid wrote what would become for centuries the reference in geometry: *the Elements*, that states the axiomatic of Euclidean geometry. At that time, even if shapes were described in a static way, some constructions were already dynamic. One can cite for instance the famous algorithm devised by Archimedes to compute π , by sequentially approximating the unit circle by polygons of increasing order. Geometry was then enriched through time, for instance the Islamic golden age

brought *algebraic geometry*, and later, from the 17th century on, *analytical geometry*, calculus and topology arose. However, geometry remained intrinsically static, either explicitly or implicitly.

The dynamic revolution of geometry came from fractal constructions in the late 19th century. New shapes with astonishing properties were discovered. For instance, Cantor described in [Cantor 1883] a set that is defined as a limit of sequence of sets, starting from the segment $[0, 1]$ and iteratively removing the opened middle third. Von Koch described in [von. Koch 1904] a curve continuous everywhere and nowhere differentiable also obtained as the limit of a geometrical sequence. For the first time, families of shapes are defined dynamically: a shape is iteratively replaced by another one. Following the same replacement pattern, mathematicians defines usually interesting self-similar shapes, such as the *dragon curve*, the *Sierpinski triangle*, or the *Levy C curve*.

After the advent of the computer in the 1950's, geometry also penetrates the sphere of Computer Science. Here again the distinction between static and dynamic representations, or equivalently between descriptive and procedural geometry persists. Combining *Computational Geometry* with the ideas of *Formal Grammars* as defined in 1956 by Noam Chomsky in [Chomsky 1956] (see section 2.1.1), procedural geometry appears first with the definition of *L-systems* (in section 2.1.2) and then with *shape grammars* (section 2.1.3). The rocketing computational power of computers as well as GPUs allows the *Computer Graphics* community to adopt procedural modeling (section 2.1.4), leading to very impressive geometries.

2.1.1 Chomsky's Formal Grammars

The linguist Noam Chomsky is famous for his work on formal grammars [Chomsky 1956], originally invented as a model for the English language, but that found applications in a wide variety of domains such as computer programming and compilers.

Generally speaking, a formal (string) grammar G is defined by a 4-tuple $(\mathcal{N}, \mathcal{T}, \mathcal{R}, \omega)$:

- \mathcal{N} is a set of non-terminal symbols.
- \mathcal{T} is a set of terminal symbols such that $\mathcal{N} \cap \mathcal{T} = \emptyset$.
- \mathcal{R} is a set of replacement rules such that a rule is a function of the form:

$$(\mathcal{N} \cup \mathcal{T})^* \mathcal{N} (\mathcal{N} \cup \mathcal{T})^* \rightarrow (\mathcal{N} \cup \mathcal{T})^*$$

where $*$ refers to the Kleene operator.

- $\omega \in \mathcal{N}$ is a start symbol or *axiom*.

Starting from the axiom ω , rules are sequentially applied (one-at-a-time) on the current string, replacing the pattern lying on the left-hand side (LHS) of the rule by the pattern lying on the right-hand side (RHS) of the rule until the string is only made of terminal symbols, and therefore no rule can be applied any longer. Such a string is called a *sentence*. This sequential replacement process

is called the *derivation process*. It might be that several rules are applicable on a given string, and therefore the choice of a specific rule will impact on the final sentence.

The *language* $L(G)$ of a grammar G is the set of all the sentences that can be obtained by deriving the grammar G in a finite number of steps. We usually distinguish three classes or formal grammars, context-free, context-sensitive, and regular grammars.

Context-free grammar (CFG) is the most famous type of grammars defined by Chomsky. The rules are restricted to the following form:

$$\mathcal{N} \rightarrow (\mathcal{N} \cup \mathcal{T})^* \quad (2.1)$$

In other words, in a context-free grammar the only rules allowed, are the ones for which the LHS is always a *single* non-terminal symbol. As a consequence, the derivation of a context-free grammar can be represented by a *derivation tree* or *parse tree*, in which the internal nodes are non-terminal symbols and the leaf nodes are terminal symbols. A very classic example of context-free grammar as well as a possible derivation tree, rooted at the axiom shape, is given in Fig.2.1.

Context-sensitive grammar (CSG) is another kind of grammar that is more generic, since the rules allowed have the following form:

$$\alpha A \beta \rightarrow \alpha B \beta \quad (2.2)$$

where $A \in \mathcal{N}$ and $\alpha, \beta, B \in (\mathcal{N} \cup \mathcal{T})^*$. α and β define the context of A , in which the rule can be applied. A parse tree can also be defined on context-sensitive grammars.

Regular grammar (RG) is a specific case of context-free grammar, where the RHS is restricted to be either the empty string (usually noted ϵ), or a single terminal symbol, or a terminal symbol followed by a non-terminal one. As a consequence, the parse tree of a regular grammar is an unbalanced binary tree, similar to a linked list structure.

Chomsky Normal Form (CNF) is a way to represent any CFG, by replacing complex replacement rules by binary rules. In other words, a grammar is in said to be in Chomsky Normal Form is all its rules are of the form:

$$N \rightarrow AB \quad (2.3)$$

$$N \rightarrow t \quad (2.4)$$

where N, A and B are non-terminal symbols and t is a terminal symbol. As a consequence, the parse tree of a grammar in CNF is a binary tree. It is important to notice that any CFG G can be formulated in a Chomsky normal form \tilde{G} , which means that the languages $L(G)$ and $L(\tilde{G})$ are identical. Chomsky normal forms turn out to be very convenient in parsing tasks.

The formal grammars, as defined by Noam Chomsky in the 1950's, form the basis upon which L-systems will be developed in the late 1960's and shape grammars then in the early 1970's.

$$\mathcal{N} = \{S\}, \quad \mathcal{T} = \{a, b\}, \quad \omega = S$$

$$\mathcal{R} = \{S \rightarrow aSb, \quad S \rightarrow ab\}$$

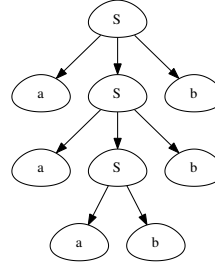


Figure 2.1: Example of context-free grammar and a derivation tree (also called parse tree). This grammar generates the language $\{a^n b^n \mid n \in \mathbb{N}\}$.

2.1.2 L-Systems

A L-system is a parallel rewriting system close to formal grammar which has been defined by the Hungarian biologist Aristid Lindenmayer in 1968 in [Lindenmayer 1968] for cellular evolution and plant growth modeling. L-systems can be used to define exact similar fractals as Von Koch's snowflake for instance.

Definition A L-system is also defined by a quadruplet $(\mathcal{N}, \mathcal{T}, \mathcal{R}, \omega)$ like a formal grammar. The only difference lies in the derivation process. Rather than applying one rule at a time, at a given time step, rules are applied simultaneously on all the non-terminal symbols of the current string. Thus, this parallel replacement process is indeed meaningful to model a growth process: for example the growth of the organs of a plant or a biological system is genuinely parallel.

At iteration t , the string produced by the L-system can be translated into a geometrical shape using a LOGO turtle paradigm. Each symbol is associated to a command of an imaginary turtle. While moving, the turtle draws the shape. Basically, some symbols are making the turtle move forward, others are making it rotate.

Example: Von Koch's curve The famous Von Koch's 2D fractal curve can easily be described by a L-system:

$$\omega = \mathbf{F}$$

$$\mathbf{F} \rightarrow \mathbf{F} + \mathbf{F} - -\mathbf{F} + \mathbf{F}$$

where the terminal symbols $\{+, -\}$ represent turtle rotations (here chosen as $+80^\circ$ and -80°), and the terminal symbol \mathbf{F} represents the turtle's moving forward command. Fig.2.2 shows different steps of the fractal curve.

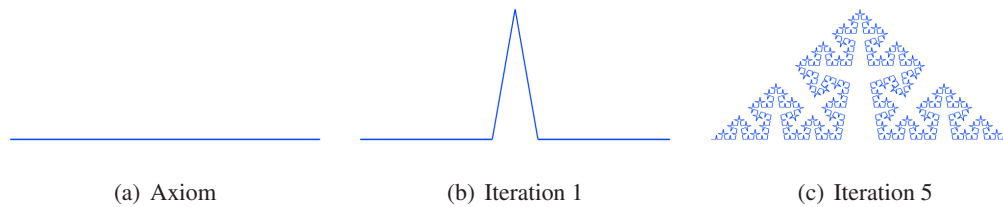


Figure 2.2: A L-system for Von Koch's curve.

Plant growth modeling There exist many extensions of L-systems, such as stochastic, parametric or context-sensitive L-systems. Parametric L-systems for instance, allow the symbols and the rules to be associated with parameters for more realistic modeling. These models have been used in Computer Graphics with great success. State-of-the-art examples can be found among others in [Prunskiewicz 1996].

Urban Modeling Using these powerful tools, initially invented to model plants, other kind of real-world shapes can be represented, especially in urban environment. The first natural extension is street network. In [Parish 2001], Parish and Müller propose an extended L-system that grows streets and street segments like branches in a tree. This L-systems takes into account external constraints such as population densities and potential city patterns such as checkerboard in New York City or radial organization around squares in Paris. A similar work has been proposed by [Coelho 2007]. They integrate geospatial data into the growth process of the L-system for the same application. Even though L-systems are very appealing for street network growth and generation, they were recently substituted by other techniques based on vector fields [Aliaga 2008] and tensor fields for graph generation [Chen 2008, Weber 2009].

Not only do L-systems have been proposed for street networks, but closer to our interest, they have been used for building modeling itself. In [Parish 2001], a parametric stochastic L-system controls the mass model of the building. A similar idea was used by [Vanegas 2010] in their Manhattan Building Grammar, used for image-based reconstruction of textured mass models of buildings. The most advanced work in building modeling using L-systems is probably the FL-system proposed by [Marvie 2005]. The nature of buildings forces the authors to extend the L-systems in different ways: they turn the terminal symbols into functions, and redefine the turtle interpretation of symbols.

According to the literature, L-systems are very efficient to model growing structure such as plants, and even though they can be modified in order to represent buildings, such geometries are not well adapted to the L-system spirit. Buildings are not naturally growing, but do show several layers of partitions (a facade contains floors, that contain windows, that hold balconies, etc.). Therefore, other grammar-like frameworks, such as the *Shape Grammars*, are better suited to represent them.

2.1.3 Shape Grammars

In the literature, the term *shape grammar* is often misused or ambiguous. In this section, we define shape grammars as they were originally thought in the early 1970's. In section 2.1.4 we will discuss more about related definitions that are often considered are part of the shape grammar framework, but for which the term *procedural modeling* is better suited.

Definition Shape grammar is a formalism to represent procedural geometry that was first introduced in 1972 by George Stiny and James Gips in [Stiny 1972] and later formalized by George Stiny in [Stiny 1980]. A shape grammar is defined as a quadruplet:

- \mathcal{V} : a (finite) set of shapes (called vocabulary)
- \mathcal{L} : a (finite) set of symbols (called labels)
- \mathcal{R} : a (finite) set of shape rules in the form $\alpha \rightarrow \beta$, where α and β are labeled shapes.
- ω : the axiom labeled shape

A rule $\alpha \rightarrow \beta$ applies to a given shape γ if there exists a transformation τ such that $\tau(\alpha)$ is contained in γ . A shape is made of (labeled) line segments. Therefore a partial order, an equality between shapes and transformations of shapes are all defined on these segments. Different classes of transformations give different levels of constraints in the shape grammar. After applying a rule, the output shape is obtained by replacing in the shape γ , the subshape $\tau(\alpha)$ by $\tau(\beta)$. Similarly, the notion of language of a grammar is defined like in formal grammars. Moreover, exactly like in the L-system formalism, the extension to parametric shape grammars is straightforward.

Example Fig.2.3 shows a classic example of a shape grammar introduced in [Knight 1994] to describe the Greek cross church plans. This is probably the most famous examples of shape grammars. It contains a single rule, transforming a rectangle into two perpendicular ones. Note that some unexpected rectangles are emerging after application of this very simple rule.

Challenges There are several challenges linked to shape grammars. In [Gips 1999], James Gips distinguishes 3 main problems related to shape grammars: automatic *generation*, *parsing*, and *inference*. In the first challenge, the main difficulty lies in the subshape detection. Indeed, it is not always trivial to detect a subshape in a shape, and unexpected subshapes often emerge after the application of a rule. There might be many such subshapes, and many rules can be applied to each of them. This makes their use particularly delicate in practice. Parsing a shape grammar consist in deciding if a given shape belongs to a language defined by a grammar, and if it does, what is the particular sequence of rules that explains this shape. Ultimately, the grammar inference is an ever more complex problem: given a set of shapes, construct a grammar generating these shapes, as well as other shapes that look alike.

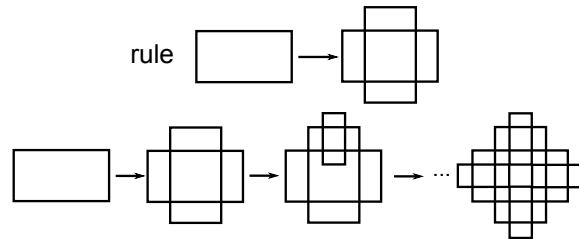


Figure 2.3: A shape grammar for greek cross church plans. The first row shows the only rule of the shape grammar. The second row shows a possible derivation starting from a rectangle as axiom. However, many more derivations are possible, including further deriving the obtained shape.

The shape grammar community has mainly proposed shape grammars that describe languages of specific interest in architecture, planning and object design. The 3 problems defined by Gips still remain open questions. However, the lack of answers to these questions (especially parsing and inference), can be explained by the very complex tasks of subshape detection.

Shape grammars for environment and planning The shape grammar community has mainly demonstrated the power of shape grammars by proposing examples of rules that generate specific classes of shapes: floor plans for Palladian villas [Stiny 1978], Turkish traditional houses [Cagdas 1996], Queen Anne style houses [Flemming 1987], Japanese tearooms [Knight 1981] and many others. But shape grammars have also proved their efficiency in industrial object design. Beyond the examples from the car industry of Harley-Davidson motorcycles [Pugliese 2002] and Buick cars [McCormack 2004], the famous Coffee Maker grammar [Agarwal 1998] has demonstrated the potential use of shape grammar on an industrial scale.

Limitations As mentioned earlier, the main limitation of shape grammars is the difficulty to match subshapes, and choose which one to process. This makes it practically useless in Computer Graphics, and that is why simplifications have been proposed to cope with the complexity of the shape grammar problem.

2.1.4 Set Grammars in Computer Graphics

Shape grammars fail at efficiently generating buildings for the aforementioned reasons. They have been restricted to little attractive shape classes for the computer graphics community. However, the concept of dynamically producing shapes is still very appealing for graphic designers because of the various impacts it could have on the whole 3D industry. If a set of semantic-geometric rules is able to catch the specificity of an architectural style and to generate on-the-fly a random model of the same style, then generating a whole city should be as difficult as generating a single building. Procedural modeling scales very well, at least theoretically. Note that this very nice property has

a cost: the one of designing the grammar that often turns out to be very sophisticated, and it also requires a procedural engine to interpret the shape grammar and derive it to produce a 3D shape.

Split grammar

In 2003, Peter Wonka decides in [Wonka 2003] to take back shape grammars into consideration, and digs out the early work from Stiny in [Stiny 1982] and the concept of *Set Grammar* some 20 years later. Set grammars consider shapes as sets of labeled basic shapes that act as symbols. This trick actually gets rid of the shape matching procedure in the derivation process. Thus, a rule can be applied to a basic shape merely if its symbol matches the symbol of the left-hand side of the rule. Set grammars are definitely better suited to efficient computer implementation of shape grammars. They can be considered as a trade-off between formal grammars and shape grammars. For that matter, in [Wonka 2003] the authors propose a specific set grammar they call a *Split Grammar*. In such a grammar, shapes can only be manipulated by carving them along an axis into several shapes. In this work, the goal was to generate random buildings using split grammars. This stochastic requirement increases the complexity of the building generation. Indeed, one could imagine to have several split rules to turn a basic shape *floor* into a sequence of basic shapes *wall* and *window*, and sample from these rules once deriving a floor, but this would very likely lead to inconsistent buildings. Thus, in this work, a control grammar is added so as to guarantee consistent choices of rule parameters. The instant architecture proposed by Wonka is not graphically perfect yet, but for the first time a shape grammar (indeed a set grammar) is used to efficiently generate random 3D buildings. The work from Wonka really initiated a procedural wave. Procedural engines as well as specific examples of grammars for any imaginable architectural style have recently mushroomed.

CGA and CityEngine

In [Müller 2006b], Pascal Müller extends the split grammar and defines a context-sensitive set grammar called CGA (that is somewhat abusively qualified as shape grammar). This work clearly defines the fundamental concepts of “shape grammar” for 3D modeling of architectural buildings. It first defines scopes, as oriented bounding boxes of the basic shapes. Then CGA supports many classes of rules (that we will later call *operators*) such as: scaling, rotation, translation, split, repeat, roof, and component split. This last operator turns a 3D mesh into a set of 2D faces, and allows the designer to efficiently go from mass modeling to facade modeling. Moreover, CGA tackles the consistency problem by defining *snap lines* and using a specific derivation order. Like magnets, the snap lines attract toward them the surrounding elements in order to enforce global alignments. The derivation process is a modified broad-first traversal of the derivation tree, where the children are visited and processed according to a priority level. This grammar is powerful enough to represent many types of architectural styles at very large scales, while being quite compact and tractable for a graphic designer. Not surprisingly, it has met a great academic and industrial success and is now a commercial product [Müller]. Antique Roman houses [Müller 2005, Dylla 2009] and Mayan architecture [Müller 2006a] have been chosen to demonstrate the capabilities of this procedural engine.

Generative Modeling Language (GML)

CGA [Müller 2006b] is a powerful tool to generate buildings, but its modeling capacities are also limited since predefined 3D meshes are sometimes required to generate fine level of details. For that matter, GML [Havemann 2004] offers a more complete modeling tool. Other than the now classic procedural rules, subdivision surfaces and free-form meshes allow the procedural description of a wider range of shapes, and extremely detailed models. The developers of GML proved the expressive power of their procedural engine in several application works such as Gothic windows modeling [Havemann 2001, Havemann 2004] or Middle Ages castles [Mudge 2005].

Other Grammars

Aside from these most famous works, many procedural engines have been released, providing different features. The most recent ones provide very impressive 3D models, and therefore corroborate the efficiency of this modeling paradigm. In an early work, [Greuter 2003] makes use of procedurally generated cities to speed-up large-scale visualization. In [Bekins 2005, Aliaga 2008], the authors couple procedural modeling with interactive image editing, to procedurally generate realistic structures. The work from [Ganster 2007] brings a user-friendly procedural modeling environment, while [Lipp 2008] allows real-time and grammar-based interactive modifications of the 3D models. Another type of procedural generation is proposed in [Finkenzeller 2006, Finkenzeller 2008a, Finkenzeller 2008b] based on graph manipulation. In [Teoh 2009], new specific rules for ancient east Asian buildings and especially curved roof construction are released. Ultimately, [Whiting 2009] succeeded in integrating mechanical constraints to procedural modeling and applied their physically consistent framework to masonry buildings such as Middle Ages castles and abbeys, with very impressive results.

2.2 Procedural Shapes

Many procedural modeling software are nowadays available (this statement was not so true a couple of years ago). However, the main objective of this doctoral work is not procedural building generation, but rather image-based procedural modeling. The difference is not as small as it may seem. Most of the software are optimized for building generation, and modifying them (when it is possible) turns out not to be straightforward or efficient. For that matter, we made the choice to develop our own procedural engine; we call it *Centrale Procedural Architect*(CPA). This section describes it and its mechanisms: the definition of shapes and rules, the derivation process and the fundamental semantic-geometric operators, illustrated by simple examples. In the remainder, we will consciously misuse the expression *shape grammars* to denote the set grammars used in procedural modeling.

2.2.1 Definitions

Shape grammars manipulate shapes and their relationships through semantic-geometric rules defined on template shapes (called basic shapes). We explain here the basic concepts to define and use shape grammars.

Basic Shapes

In our framework, we call *basic shape* a geometric primitive (mesh) defined in the unit cube, associated with appearance attributes (color or texture), and a symbol that we call *semantic*. In the notations we are using here to represent the grammar rule, the semantic is basically the name of the basic shape. Basic shapes can be terminal or non-terminal, depending on their semantics only. The primitive can be a cube, a cylinder, or any kind of mesh. Basic shapes are indeed abstractions, as template of shape that can be instantiated in the 3D world through a *scope*.

Scopes

A scope is basically an oriented bounding box. It can equivalently be seen as a 3D transformation, mapping the unit cube to this bounding box. This transformation is made of a 3×3 rotation matrix \mathbf{R} , a translation \mathbf{t} and a 3×3 non-uniform scaling matrix $\mathbf{\Sigma}$, that represent 3 scalings $\sigma_x, \sigma_y, \sigma_z$. Thus a scope S is a map:

$$\begin{aligned} S : \mathbb{R}^3 &\rightarrow \mathbb{R}^3 \\ X &\mapsto \mathbf{R}\mathbf{\Sigma}X + \mathbf{t} \end{aligned} \quad (2.5)$$

Using homogeneous coordinates, this mapping is formulated as matrix products:

$$X \mapsto \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix} \begin{pmatrix} X \\ 1 \end{pmatrix} \quad (2.6)$$

We can define an inner composition rule among non-degenerated scopes (the scalings are all non-zeros). Two scopes $S_1 = (\mathbf{R}_1, \mathbf{t}_1, \mathbf{\Sigma}_1)$ and $S_2 = (\mathbf{R}_2, \mathbf{t}_2, \mathbf{\Sigma}_2)$ can be naturally composed in the following way:

$$S_1 \circ S_2 = (\mathbf{R}_1 \mathbf{R}_2, \mathbf{t}_1 + \mathbf{t}_2, \mathbf{\Sigma}_1 \mathbf{\Sigma}_2) \quad (2.7)$$

Provided that the scaling is non-degenerated, the scope transformation is invertible. The inverse of a scope is also a scope given by:

$$\begin{aligned} S^{-1} : \mathbb{R}^3 &\rightarrow \mathbb{R}^3 \\ X &\mapsto \mathbf{\Sigma}^{-1} \mathbf{R}^T (X - \mathbf{t}) \end{aligned} \quad (2.8)$$

The identity element of the group of non-degenerated scopes is $I_S = (\mathbf{I}_3, \mathbf{0}, \mathbf{I}_3)$.

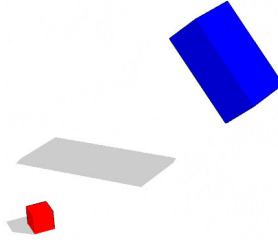


Figure 2.4: The scope is a transformation of the 3D spaces that positions and resizes objects in the 3D world. The red unit cube is mapped to the blue rectangular cuboid through the scope transformation.

Atomic Shapes

We call atomic shape the association of a basic shape and a scope. It can be thought as an instantiated basic shape, since it has a well-defined position, dimension and orientation in the 3D world. An atomic shape has a local coordinate system defined by its scope. A shape is a collection of atomic shapes, sequentially built by a grammar derivation process that we will define soon.

Replacement rule

A rule is denoted *precondition* : $LHS \rightarrow RHS$, and maps an atomic shape called *LHS* to a collection of atomic shapes called *RHS* under a (possibly empty) precondition. A rule can be applied on an atomic shape a of a shape s if the *precondition* is verified and if the semantics of *LHS* and a are identical. Then the atomic shape a is replaced by the atomic shapes in *RHS* that we denote RHS_1, \dots, RHS_n . However, their scopes have to be recomputed since *LHS* and a do not necessarily share the same positions, orientations and dimensions.

Shape grammar

From now on we redefine a *shape grammar* as a quadruplet $(\mathcal{N}, \mathcal{T}, \mathcal{R}, \omega)$ where:

- \mathcal{N} is a finite set of non-terminal basic shapes
- \mathcal{T} is a finite set of terminal basic shapes
- \mathcal{R} is a finite set of rules. We denote a rule: *precondition* : $LHS \rightarrow RHS$
- ω is an atomic shape called *axiom*.

If the precondition only tests the properties of a , then the grammar remains context-free. However, nothing prevents the precondition from testing the whole shape s or a subshape that contains a , for which case the grammar is said to be context-sensitive.

Algorithm 2.1 Derivation Process

```

s = axiom
while a != NULL do
  rule = sampleRule(s)
  apply(rule,s)
  s = s.next
end while

```

2.2.2 Derivation process

Depth First Search Scheme

Unlike Pascal Müller's grammar, the derivation process does not follow a Broad First Search traversal (BFS), but a Depth First Search (DFS) one. This property will be exploited while parsing the grammar (see chapter 4). For the time being, this is a mere convention that is both efficient and natural for computer implementation.

Starting from the axiom, a rule is sampled from the applicable ones and applied. The new born atomic shapes are recursively processed one at time, until no rules are applicable anymore. As mentioned in section 2.1.1, the derivation process can be represented by a derivation tree. Applying a rule to an atomic shape is equivalent to adding children to the corresponding node in the derivation tree. A shape is fully described by its derivation tree and the concepts of BFS and DFS traversal make complete sense while working with tree data structures.

Inheritance

The rules may not specify completely the attributes of the atomic shapes. As a consequence, we follow the convention that if the rule does not specify some attributes, they are transferred from the parent to the children. For instance the new atomic shapes can inherit texture or color parameters as well as *tags* (see in section 2.3).

2.2.3 Operators

In order to simplify the definition of rules, we define *operators*. They are simple procedures that act as factories of atomic shapes, based on a reduced set of parameters. An operator takes as input an atomic shape and returns atomic shapes. The number of offspring clearly depends on the operator, its parameters and the atomic shape it is applied on. We call *stem shape* an operator or a basic shape, since both can produce atomic shapes. We will denote the operators in the following way:

$$\text{Operator}(\text{stems}:\text{parameters})(\text{LHS}) = (\text{RHS}_1, \dots, \text{RHS}_n)$$

It means that an operator is defined by some parameters and some stem shapes. While applied on an atomic shape called LHS it generates a set of atomic shapes RHS₁ to RHS_n. As said

earlier, stem shapes are either basic shapes, or operators. This nesting property is very convenient, since it enables us to define only the elementary operations and compose them as much as needed. The operator notations that we give in this section are only used to facilitate the representation of the rules on the paper. In practice, rules, operators, basic shapes and parameters are represented as XML elements in an XML file that describes the shape grammar.

What is the relationship between rules and operators? Basically in our grammar, rules are going to be defined as a collection of operators (usually a single one, but potentially as many as necessary). During the application of a rule, an atomic shape a is replaced by other atomic shapes. The operators of the rule compute from a the new atomic shapes a_1, \dots, a_n , which are then inserted in the derivation tree as children of a . From now on, we will refer to a as the parent, and to a_1, \dots, a_n as the children.

Numerical Parameters

The operators usually take numerical parameters that represent distances, angles, numbers or anything meaningful. These parameters can be of three types: *absolute*, *relative*, or *cardinal*. Absolute and relative parameters affect distances. Unlike absolute parameters, relative ones are defined relatively to the scope of the LHS. Cardinal parameters represent integers. To distinguish between them in our notations, we add the suffix *a* for absolute, *r* for relative, and *c* for cardinal. For example 3_r means “3 relative”.

On top of this distinction, parameters can be defined statically or dynamically. We call the first ones *literals* and the second ones *expressions*. Expressions are evaluated on-the-fly at runtime depending on the LHS they are applied on, whereas literals are fixed values defined out of any context. From an implementation point of view, the evaluation of any kind of expressions is possible using an embedded script language (python in our case).

3D Transformations

The most natural way to manipulate shapes is to modify their scopes. Since a scope is a composition of 3 standard transformations, they all define an operator: rotation, translation and scaling.

Translation The *translate* operator takes as parameters one of the 3 *local* axis X , Y or Z of the parent scope, a scalar parameter t and a single stem shape. The numerical parameter represents either the absolute or the relative length of the translation with respect to the dimension of the parent scope in the axis direction. We represent a translation along the axis ax as:

$$\text{Translate}_{ax}(\text{stem}:t)$$

Rotation The *rotate* operator takes a local axis, an angle, a rotation center, defined in the local coordinates system of the parent in the plan orthogonal to the rotation axis and a single stem shape. Omitting the rotation center, we represent a rotation along the axis ax as:

$$\text{Rotate}_{ax}(\text{stem}:angle)$$

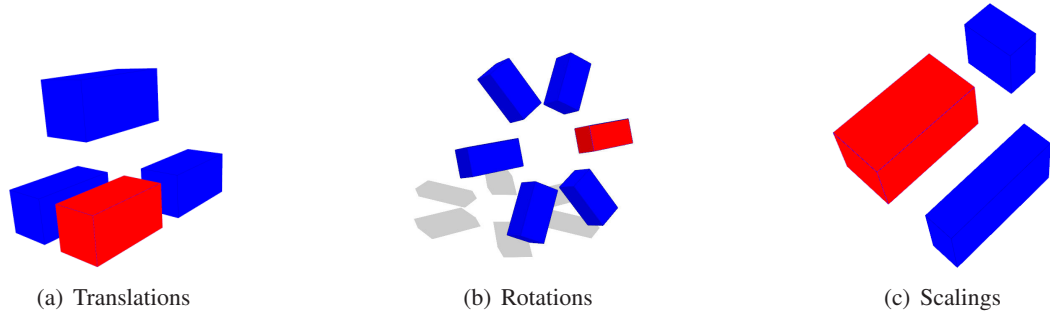


Figure 2.5: 3D Transformation operators. The red shape are the LHS and the blue ones are several RHS, after applying several times the operators, with different sets of parameters. In these examples, the basic shapes' semantics are represented by the two colors, and the geometric primitives involved are always cubes. After scope transformations, the cubes are turned into cuboids.

Scaling Similarly, the *scale* operator modifies the scale of the current scope along a direction given as parameter. The rescaling can be relative or absolute, and an additional parameter specifies the center of the scaling (defined in the parent scope coordinate system). We represent a scaling along the axis ax as:

$$\text{Scale}_{ax}(\text{stem} : s)$$

Reflection Sometimes, applying reflection turns out to be very convenient, since many shapes have symmetry axis. However, the definition of reflection requires a modification of the scope definition. As a consequence, we augment the scope with 3 symmetry values, which can be 1 if no reflection has been applied on this direction and -1 otherwise. We represent a reflection with respect to the axis ax as:

$$\text{Reflect}_{ax}(\text{stem})$$

Splitting operator

Splitting operators were introduced in [Wonka 2003] for architectural needs. Indeed these operators are very useful. We define three types of splits, all carving the scope of the LHS along one of its principal axis into several chunks.

Split Giving a local axis (X , Y or Z), a set of lengths $\{x_i\}_i$ (relative or absolute) and set of stem shapes, the *LHS* is carved into several chunks of respecting lengths. If we denote x the length of scope of LHS in the axis direction, then we see that several problems may arise. What to do if $x < \sum_i x_i$ or $x > \sum_i x_i$? There exist indeed several modes for this fundamental operator, that correspond to different policies to handle these specific cases and conserving either the ratio or the absolute dimensions of the shapes.

First, we define the number of shapes that fit in the scope of the LHS as:

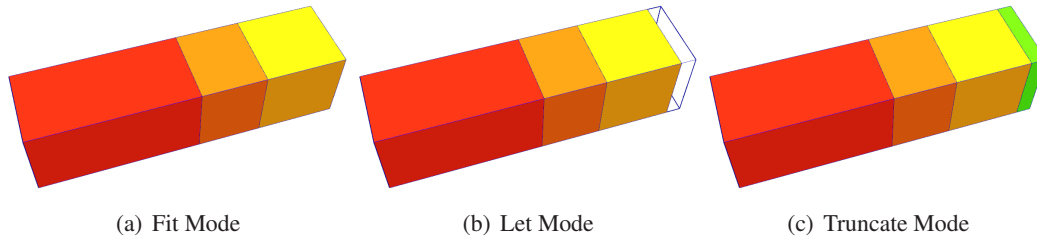


Figure 2.6: Split operators. This example shows how to split of an atomic shape of size 40 along X with 4 basic shapes, of size 20, 8, 10 and 16. In (a), the 3 first shapes are kept, and resized so that they fit the parent scope. In (b), the 3 first shape are kept, and not resized, therefore an empty space remains on the right of the yellow shape. In (c), the green shape does appear but is truncated.

$$k = \max_j \sum_{i=1}^j (x_i) < x \quad (2.9)$$

In the *fit* mode, we keep the k first shapes. If $\sum_{i=1}^k x_i < x$, the scopes of the k first atomic shapes are rescaled by a factor $\alpha = x / \sum_{i=1}^k x_i$, so that the proportions between the shapes remain unchanged and the children shapes still fill entirely the scope of the *LHS*. As a consequence, the final dimensions of the atomic shapes are not exactly the ones that were given as input. This property is sometimes annoying, and therefore we define two other kinds of modes. In the *let* mode, the k first shapes are kept as well, and the remaining ones are also discarded. However, unlike in the fit mode, the atomic shapes are not rescaled. Therefore, an empty space is left at the end of the split and the volume of the parent scope is not conserved (see Fig.2.6(b)). The empty space is very often unwanted. For that matter, the *truncate* mode is a trade-off between the two first ones. In this mode, the shapes are added until they completely fill the parent scope. The last one usually does not have the size it was expected to have, but simply fills the empty space that would have been left in the *let* mode. However, all the other atomic shapes have the same dimensions as specified by their parameters. The 3 split modes are shown in Fig.2.6. Omitting the mode to keep the notation understandable, we represent the split operator as:

$$\text{Split}_{ax}(\text{stem}_1 : x_1, \dots, \text{stem}_n : x_n)$$

Repeat The *repeat* operator was first introduced by Pascal Müller in [Müller 2006b]. It is indeed a regular split of the same element. For architecture modeling, this operator is indeed very convenient, since most of the time architectural elements get repeated regularly over the facade. Repeat also takes an axis as input, a single stem shape and a parameter that can be a cardinal or an absolute value. An absolute value corresponds to the length of the element to be repeated in the axis direction. This length is actually adapted so as to fit an integer number of elements. A cardinal value n corresponds to the number of elements to be repeated. The size of each element is basically $\frac{x}{n}$ where x is the length of the scope in the axis direction. Repeat operator is represented as:

$$\text{Repeat}_{ax}(\text{stem} : x)$$

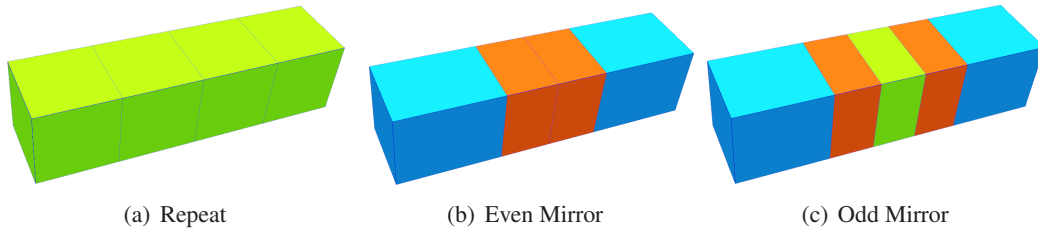


Figure 2.7: Repeat and Mirror splits. The mirror splits are nested with split operators in order to better visualize the symmetry.

Mirror Split A *mirror* split enables the designer to introduce symmetries, and therefore to work on half of the model. The descendants of the reflected atomic shapes keep the symmetry as long as no *reflect* or *mirror* operators are not applied. The mirror operator has two modes: *even* and *odd*. The even mode carves the parent scope into two identical parts along the input axis. The left half receives the stem shape given as input, and the right one a reflected version of it. In the odd mode, the parent scope is cut into three parts, two symmetric ones, and a middle part. As a consequence, the user has to provide two stem shapes and two numerical parameters. Therefore, the mirror split creates either two or three atomic shapes depending on its mode. As a consequence, it is often combined with a nested split operator. The two versions of the mirror split are represented as:

$$\text{Mirror}_{ax}^{\text{even}}(\text{stem}) \text{ and } \text{Mirror}_{ax}^{\text{odd}}(\text{mid}:x_1, \text{sym}:x_2)$$

2D-3D operators

Two operators deal with 2D-3D transformations. They are very useful in building modeling as we will see in section 2.3, and they turn out to be quite convenient for general modeling purposes.

Extrusion An *extrusion* enables the designer to go from 2D shapes to 3D ones. Unlike the previous operators that act on scopes only, the extrude operator acts on the primitive itself (the mesh) as well as its scope. If the mesh is made of a single face, then an extrusion of a height h will turn this face into a generalized cylinder of height 1 since the scope is always defined in the unit cube. The scope will have a height of h . An extrusion takes as input a LHS, a semantic and a numerical parameter, and returns a single atomic shape. We represent the extrusion operator as:

$$\text{Extrude}(\text{stem}:h)$$

Facetization This operator was also introduced by Pascal Müller in [Müller 2006b] under the name *Component Split*. It is actually a key element in building modeling since it enables to go from mass modeling to facade modeling. This operator turns out to be useful in many other situations. Like the extrusion, this operator acts on the mesh and the scope. The idea of `Facetize` is to turn a mesh into a set of single face meshes, each one being the primitive of an atomic shape, whose scope is computed to best fit the face. The normal to the face correspond to the Z axis of the new scope. The X axis is computed using the first edge of the face polygon, and the Y axis is simply the cross product of Z and X . If the faces have a predefined label, such as *street* or *courtyard*, *neighbor*, *top* or *bottom*, then the operator offers the possibility to filter out the output with labels

```

 $\omega$  → step +
      Translationz(Rotatez( $\omega:\alpha$ ):1r)

```



Figure 2.8: Stairs grammar example. On the left the single-rule-shape grammar to make a step. On the right the shape obtained after 25 iterations. The red shape represents the axiom, the blue ones represent the “steps”.

given in parameters. This is convenient to design separately the different facades of the building, depending on their facing the street or not. We represent it as:

```
Facetize(stem:label, ..., stem:label)
```

Colorize

We propose a *Colorize* operator that takes as input an LHS, a stem shape and a numerical parameter (usually an expression). The value of the parameter is evaluated at run-time, and sets the color of the shape through its hue value in HSV color space. This operator is quite useful to produce visually nice effects. In Fig.2.9 and Fig.2.8 for instance, we omitted in the grammar the colorize operator, however it was used in practice to obtain nice visual effects based on the positions of the atomic shapes. Note that having many different meshes associated to many different materials is not efficient for rendering speed. In practice, we gather the meshes according to their materials and create aggregate meshes that simplify the scene graph, and that can be sent once and for all to the graphic card. Many different materials imply a bigger scene graph, and therefore worse rendering performances. We represent it as:

```
Colorize(stem:expression)
```

Logical operators

All the operators presented above modify the attributes of the LHS they are applied on: either the scope, the mesh, or the appearance. Here we introduce two operators that enables the designer to choose between several operators or to concatenate several ones.

Switch The *switch* operator acts like many preconditions. It takes as input a LHS atomic shape, a set of Boolean expressions and a list of stem shapes. Each expression is associated to a stem shape. The operator chooses the first stem shape for which the expression is correctly evaluated. The condition depends on the attributes of the LHS, either geometric, semantic, etc. We denote it as:

```
Switch(stem:expression, ..., stem:expression)
```

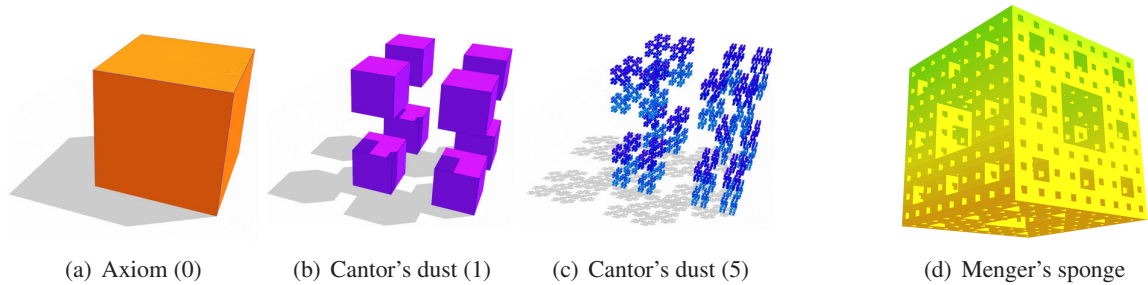


Figure 2.9: Cantor's dust and Menger's sponge shape grammars.

Union The union operator is very simple. It takes a list of stem shapes and concatenates them all. This is particularly useful when dealing with operators that are expecting a single RHS such as scaling. We note it:

`union(stem, ..., stem)` or simply `stem+...+stem`

2.2.4 Simple Examples

Here we give some simple examples of shape grammars, defined using the aforementioned operators, so as to get a sense of how to combine simple procedures to generate complex structures. Note the compactness of these grammars compared to the visual complexity of the generated models.

Stairways Stairs are often used as a metaphor to teach induction principle or recursion. Here, we propose a very simple recursive shape grammar to build stairs, “step by step”. This grammar is made of a single rule; the axiom is replaced by a basic shape we call `step`, that inherits the geometry and scope from the axiom ω and a rotated and translated version of the axiom. The stairs grammar as well as an procedurally generated stairway are given in Fig.2.8.

Cantor's dust The Cantor's dust is a 3D generalization of the Cantor's fractal set [Cantor 1883]. A cube is replaced by 8 cubes, located at the 8 vertices, included inside the parent cube, and with an edge three times smaller. Here is a representation of a shape grammar to generate it. Fig.2.9(a,b,c) shows 3 steps of its construction.

$$\begin{aligned} \omega &= X = \text{Cube} \\ X &\rightarrow \text{Split}_X(A:1/3r, \emptyset:1/3r, A:1/3r) \\ A &\rightarrow \text{Split}_Y(B:1/3r, \emptyset:1/3r, B:1/3r) \\ B &\rightarrow \text{Split}_Y(X:1/3r, \emptyset:1/3r, X:1/3r) \end{aligned}$$

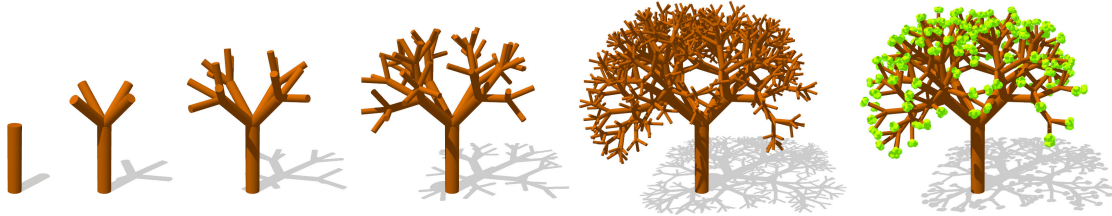


Figure 2.10: Generation of a tree using a shape grammar.

Menger's sponge Here are the rules to recursively apply on a cube to obtain the Menger's sponge. The basic rule removes the middle 3D cross from the cube. In this example, we decompose this complex rule using the presented operators. We could also have used nesting to make it a single rule. Fig.2.9(d) shows the fourth iteration of the construction of the sponge.

$$\begin{aligned} \omega &= X = \text{Cube} \\ X &\rightarrow \text{Split}_X(A:1/3r, B:1/3r, A:1/3r) \\ A &\rightarrow \text{Split}_Y(C:1/3r, D:1/3r, C:1/3r) \\ B &\rightarrow \text{Split}_Y(D:1/3r, \emptyset:1/3r, D:1/3r) \\ C &\rightarrow \text{Split}_Z(X:1/3r, X:1/3r, X:1/3r) \\ D &\rightarrow \text{Split}_Z(X:1/3r, \emptyset:1/3r, X:1/3r) \end{aligned}$$

Trees Simple tree structures can be represented by a very compact grammar. The idea is the following: a bud gives birth to a branch, and a branch gives birth to a trunk and three or four rotated buds. To make it more realistic, we turn the angles of the rotation into random variables. Different steps of the construction of a tree are given in Fig.2.10. Random angles and randomly chosen rules (3 or 4 buds) confer a quite realistic geometry to the tree. In general, I think that too regular grammars perform badly at modeling real-world geometries. Adding stochasticity usually enhances the visual impression of the shapes: perfection is not of this world! Here is a representation of this tree grammar:

$$\begin{aligned} \omega &\rightarrow \text{Scale}_Z(\text{branch}:10a) \\ \text{branch} &\rightarrow \text{trunk} + \text{bud} + \text{Rotate}_Z(\text{bud}:120a) + \text{Rotate}_Z(\text{bud}:240a) \\ \text{branch} &\rightarrow \text{trunk} + \text{bud} + \text{Rotate}_Z(\text{bud}:90a) \\ &\quad + \text{Rotate}_Z(\text{bud}:180a) + \text{Rotate}_Z(\text{bud}:270a) \\ \text{bud} &\rightarrow \text{Rotate}_X(\text{Scale}_{X,Y,Z}(\text{branch}:0.5r) : \text{random}(30a, 60a)) \\ \text{bud} &\rightarrow \text{Scale}_{X,Y,Z}(\text{leaf}:0.5a) \end{aligned}$$

So far, we have seen how to easily manipulate shapes through shape grammars. The great power of shape grammars lies in the ability to express a huge number of shapes, all sharing the same design or spirit, and in a very compact way. However, the purpose of this thesis is to target a specific class of shapes: buildings. For that matter, we are going to discuss in section 2.3 how to design them, and define additional operators to reach this goal.

2.3 Building Modeling with Shape Grammars

Buildings are complex but structured shapes. Even among an architectural style, the number of floors, of windows per floor as well as the dimensions of these architectural elements may vary from a building to another, but also may vary inside a single building. Therefore shape grammars seem to be very appropriate to describe in a compact way all these shapes, without giving an exhaustive description of all the expected configurations, but rather by giving principle rules of construction. However, a key operator is missing so far to produce buildings: the one that creates a roof. This issue is tackled in section 2.3.1.

Now, assuming that we have designed such a shape grammar, capable of describing a specific architectural style, a problem still arises because the grammar defined so far is completely context-free. Let's assume that this shape grammar has 2 different rules to split a floor into a sequence of windows and walls. Now imagine that our building has two floors. If the rules are randomly sampled according to a uniform law during the derivation process, 50% of the time, the split of the first floor will not match the split of the second floor. This configuration may happen in real-life but is very unlikely. This blind sampling is due to the context-free property of the grammar. Rules are applied only based on the current shape attributes, without taking care of what rule have been applied anywhere else before. Moreover, if we have a more realistic setting, such as 100 different rules and 5 floors, then the probability that all the floors are split the same way goes down to 10^{-8} . In other words, a random generator of buildings will produce a regular building (all the floors are split the same way) only once over 100 millions trials. This is definitely not acceptable.

For that matter, in section 2.3.2 and 2.3.2 we introduce two new operators that control the level of consistency of a shape. The proposed solution is different from the one proposed by [Wonka 2003] that uses a second grammar in charge of controlling the parameters, and also different from [Müller 2006b], that uses *snap lines* forcing the shapes to stick to them. Our solution modifies the derivation process to take into account the shape history. In section 2.3.3 we give two examples of grammars to generate real architectural buildings.

2.3.1 Roof Operators

Knowing the shape of the footprint (a polygon), how to raise a roof that fits? The solution was given by [Aichholzer 1995] and computes a straight skeleton for the polygon. Unlike the medial axis transform [Choi 1997], the straight skeleton is guaranteed to be a set of segments and that is why it has been used to raise roofs [Aichholzer 1995, Eppstein 1999]. In this work, we have implemented a weighted version of this algorithm.

Straight Skeleton Algorithm

The idea of the algorithm is to shrink the polygon by moving each edge along its normal direction at a given speed. The moving edges are colliding, and so some edges will disappear little by little, or some will be split into two. The spirit of this algorithm is to turn the problem of computing *when*

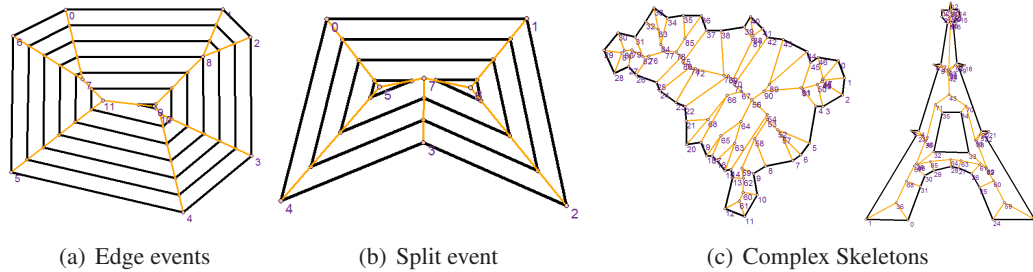


Figure 2.11: Straight Skeleton Algorithm. (a) a convex polygon for which only edge events occur. (b) a split event, that creates a second loop in the polygon. (c) complex shapes with many vertices and holes.

these events are going to happen into *where* they are located.

Basically, each vertex of the polygon moves along the bisector of the edges it is connected to. The bisector takes into account the speed (weight) of both edges. Therefore, three main events may occur. Two end vertices of a common edge may meet (edge event), a reflex vertex (concave point) may meet an edge (split event) or two or more reflex vertices might meet (reflex event). After each event, the topology of the polygon changes and therefore the bisectors change as well. The algorithm jumps from one event to another, storing the events in a priority queue, where the priority is the distance to the event. The algorithm stops when the polygon is degenerated. The straight skeleton is obtained by following the trajectories of the vertices of the original polygon. Fig.2.11 shows different steps of the straight skeleton algorithms as well as complex examples.

Raising Roofs

Using the straight skeleton algorithm, we derive two operators to generate three kinds of roofs. The first one called `Hip` can generate *hip* and *gable* roofs and the second one called `Mansard` can generate *mansard* roofs. Examples of these different types of roofs are given in Fig.2.12.

Hip operator The hip operator is a direct application of straight skeleton algorithm. It takes as input a basic shape and computes the mesh of the RHS based on the mesh of the LHS, without any parameter. The scope of the created atomic shape is actually the same as the LHS. So the idea is to raise a roof from the straight skeleton. This polygon is the *top* face of the mesh of the LHS. The goal is to compute the topology of the faces of the roof mesh, and the height of each vertex in the unit cube. For that matter, we represent the straight skeleton as a graph G , where the edges of the original polygon are *not* edges of G . To compute the faces of the roof mesh, we take all the edges of the original polygon, and compute for each one the shortest path from one end of the edge to the other using Dijkstra algorithm. This shortest path is not the edge itself, since it does not belong to G , but goes through all the vertices of the face of the roof supported by this edge. Repeating the

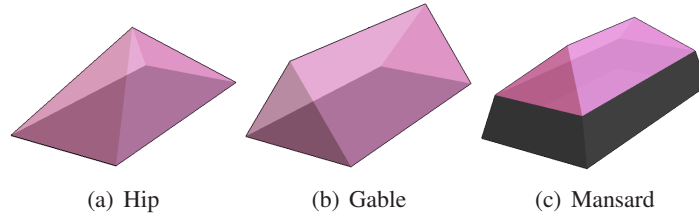


Figure 2.12: Roof operators generate (a) hip, (b) gable and (c) mansard roofs.

same procedure for each edge of the polygon gives the faces of the roof mesh.

Moreover, the highest vertex of the straight skeleton should be at altitude $z = 1$ in the unit cube. This highest point is actually the last point to have emerged in the algorithm, since in 3D, moving the edge toward the inside of polygon at a given speed is equivalent to moving the edge up, toward the interior of the polygon with a given slope. When two points collapse in the algorithm, they are indeed at the same altitude in the 3D world. The more a vertex survives in the shrinking procedure, the higher it is in the 3D world.

The difference between a gable roof and a hip roof only relies in the speed or slopes of the edges. Using the straight skeleton with a uniform speed gives hip roofs. Setting some edge speed to 0 creates gable roofs. The speed of the edges is indeed given by the label (street, courtyard and neighbor) of the original polygon. We note the hip operator:

`Hip(stem)`

Mansard Roof The mansard roof is also based on the straight skeleton algorithm. However, this roof design is a bit more complex since it has two parts. We call the lower part the *loft* and the upper part the *top*. Each part has a different slope. Usually the loft slope is stiffer than the top one. The *Mansard* operator takes two stem shapes and two parameters: the angle α of the slope of the loft, and the height of the loft h . To build a mansard roof, we abort the shrinking process of the straight skeleton algorithm at $h \sin(\alpha)$. We build the loft with this first step. Then we go until the end of the shrinking process to create the top, calling the hip roof procedure. We represent it as:

`Mansard(loft, top, angle, height)`

Shrink Operator Using the same idea, we can also create a *shrink* operator, which purpose is to modify a polygon by shrinking it (or expanding it). One could think that scaling would be as efficient as shrinking, but as soon as the polygon is not convex, no scaling can properly perform a shrink. Shrinking is particularly useful while creating stairs on non-convex shapes (see section 2.3.3). The shrink operator takes a shape with a single polygon and a shrinking distance. Thus, shrinking is often combined with a facetization or an extrusion operator. We denote it as:

`Shrink(stem:x)`

Now that we know how to raise roofs based on a polygon, we are going to introduce operators that control the level of consistency of the building by modifying the derivation process.

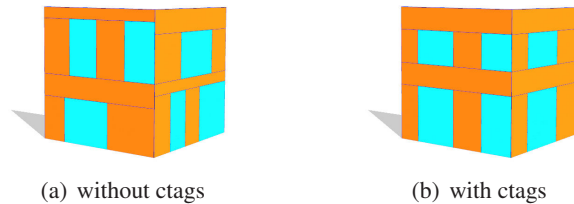


Figure 2.13: Consistency Tags. On the left, no tags are used. Notice how the floors on the two facades and the windows within each facades do not match. On the right the two facades are tagged with the same *ctag*, and the floors are tagged with a *ctag* that is the same among a facade, but different from a facade to another. Thus we obtain consistent floors, aligned windows with different layouts on the two facades.

2.3.2 Tag Operators for Consistent Shape Generation

As explained earlier, the combination of random variables and context-free grammar may not be convenient for pure generation. As a consequence we introduce here tag operators.

Consistency Tag

So far, atomic shapes were defined by a basic shape (semantic, geometry and appearance), and a scope (translation, rotation, scaling). We add to these attributes a tag called the *ctag* and which is merely a number. The value of this *ctag* is set by the `CTag` operator (for Consistency Tag) that takes as input a numerical parameter and a stem shape (basic shape or operator). After applying a `CTag` operator, the new born atomic shapes are tagged. This tag will be inherited to their descendants as long as no other `CTag` operator is applied. We note it:

$$\text{CTag}(\text{stem}; \text{tag})$$

The *ctag* has an impact on the shape design through the derivation process. The idea is very simple: apply the same rules on the same tagged atomic shapes. So far, the procedural engine was randomly choosing the rules based only on the semantic of the LHS. This procedure is still valid if the LHS is not tagged. For tagged atomic shapes, the procedural engine maintains a history of rules and *ctags* they were applied on. As a consequence, when a tagged atomic shape A is to be processed, we first check if another atomic shape A' with the same *ctag* and the same semantic has already been processed before. If such a shape A' exists, then we choose to apply the exact same rule on A that we applied on A' . If no such rule is to be found, then we randomly pick up a rule and record it in the history. The consistent derivation process is described in pseudo-code in algorithm 2.2

Let's consider the example mentioned earlier. We have two rules to split a floor and two floors to be split. If the floors are tagged with the same tag, then the derivation process will choose for the second floor, the same rule it has chosen for the first one. However, if one at least of the floors has no tag, or if their tags are different, they will be processed independently. In other words, *ctags*

Algorithm 2.2 Consistent Derivation Process

```

s = axiom
while s != s.end do
  rule = 0
  if s.ctag < 0 then
    rule = sampleRule(s)
  else
    if history[s.id,s.ctag] != NULL then
      rule = history[s.id,s.ctag]
    else
      rule = sampleRule(s)
      history[s.id,s.ctag] = rule
    end if
  end if
  apply(rule,s)
  s = s.next
end while

```

are a simple way to bind the destiny of atomic shapes, in order to impose consistency. The effect of *ctags* is demonstrated in Fig.2.13

Differentiation Tag

Consistency Tag handles the balance between stochasticity and consistency. Same shapes can be processed the same way, provided that they share the same ctag and semantic. However, there exist cases in which we would like to split the same way two atomic shapes that should not be identical. For instance, we may want to have aligned windows in the roof and on the facade. However, in the end, we do not want the roof and the facade to look alike. For that matter, we introduce another kind of tag called *dtag* and the corresponding operator: the DTag or differentiation tag. The idea is pretty simple. As long as shapes have to be processed the same way, they should share the same semantic and the same ctag. However, when it is time to differentiate them, we should have a mechanism to recognize the roof from the facade. This is exactly the purpose of the dtag. At the creation of the roof floor, it is created as a floor, with a floor semantic and a ctag equal to the other floors' tag, but the roof floor receives also a dtag that will be inherited to its descendants. Thanks to this tag, the descendants will know that they are supposed to be part of a roof. Using precondition or *switch* operators it will then be possible to differentiate between the roof shapes and the facade ones. This situation illustrated here with roof and facade may also occur in other contexts. We note this operator:

$$\text{DTag}(\text{stem}:\text{tag})$$

Now that we have defined the basic operators for building modeling, let's give some insight of how to build stochastic consistent buildings. For that matter, we propose two different examples of grammars: one for Doric Greek ancient temples, and the other one for Parisian Haussmannian buildings.

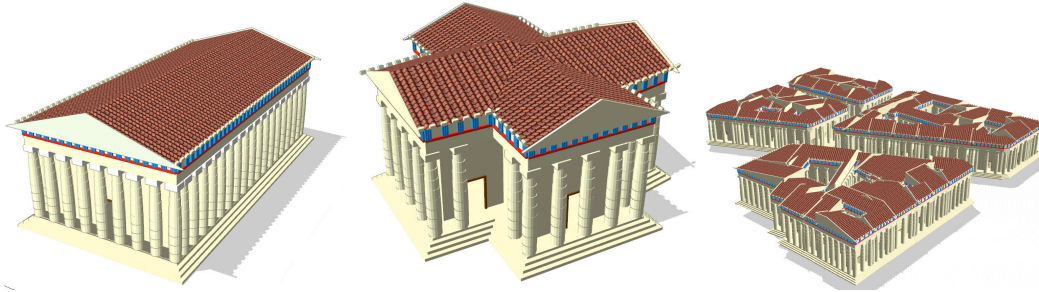


Figure 2.14: Procedural Greek Doric Temples generated on several footprints, and on a small neighborhood.

2.3.3 Examples of Shape Grammars

In this section, we provide two examples of grammars that stochastically generate a building from a footprint axiom. These grammars use all the operators defined above. For highly detailed models, we also use some predefined 3D models for basic shapes such as ornaments or doors.

Shape Grammar for Doric Temples

Ruins of Greek Doric temples have been found in a region going from Sicily and southern Italy to the Turkish coasts of Anatolia. Enough of these temples have been discovered so that geometrical rules were manually inferred by architects and specialist of the Ancient Greece. The most famous example of this architecture is the *Parthenon* in Athens. Imitating the work of classical architecture's specialists, we have designed a grammar to generate Doric-like temples. This grammar is not stochastic, but the generated 3D models may vary from one temple to another, depending on the geometry of their footprints (axiom of the grammar). These models may not be exactly following the principles of classical architecture, but are graphically satisfying. May Phidias, Callicrates and Itkinos excuse me! The grammar given at the end of this chapter in table 2.4 describes this class of shapes. It is entirely procedural; no 3D models are used to enhance the final model quality. We omitted preconditions and switch operators on purpose, to clarify the grammar and focus on the main procedures. However, these additional tests are necessary to tackle degenerated cases in practice, for example while trying to split with a parameter greater than the parent's size.

Shape Grammar for Haussmannian Buildings

In the 1860's, Paris went through the biggest mutation a city has ever lived which was not caused by a fire. The reason for this fast transformation has a name that all the Parisians know: baron George Eugene Haussmann. In charge of the city, he took advantage of the convenient legislation to entirely reshape Paris and make it a modern and healthy city. Annexing neighboring cities, drawing new streets and avenues, buildings public parks the city was metamorphosing and new

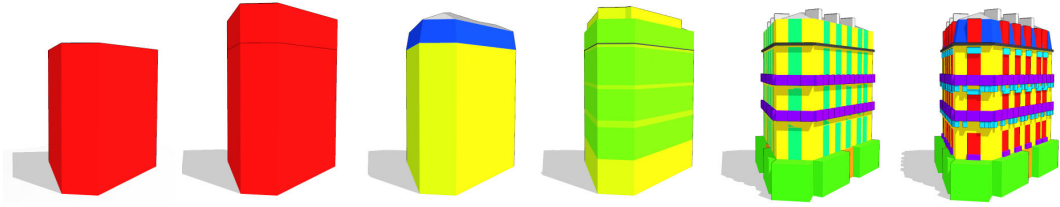


Figure 2.15: Step-by-step construction of an Haussmannian Buildings.

buildings were mushrooming everywhere. These buildings were on purpose very similar: same heights, materials, shapes, ornaments, roofs. Their uniformity created the current image of Paris [Loyer 1987]. They have four to six floors (the last one in the loft of the roof), made of dimension stones, have a running balcony on the second and fifth floors, and thin straight intruded horizontal lines on the first one. The mansard roof is covered by zinc, which gray tone perfectly matches the color of the Parisian sky. This Haussmannian style was at that time very well ruled and conformity was checked by architects on each project. This style emerged from aesthetic rules, economic and time constraints, as well as the availability of construction materials. Therefore, we are facing here a perfect example of an architectural style that can be described by grammar rules, and for which we have a huge number of instances. These buildings are all very recognizable, but still show important geometric variations. Here again a grammar is very well suited to handle this intrinsic diversity. For that matter, we designed a stochastic consistent grammar, capable of catching the variability of the possible geometries and topologies and the uniformity of the architecture. Whenever the parameter is given as a number with a unit, it means that this number is fixed. Otherwise, the parameter represents a random variable (see table 2.5 at the end of this chapter).

The idea of this grammar is surprisingly very similar to the one creating a temple. The first steps are almost identical: a footprint is extruded to create a mass model. The mass model is divided into a roof part and a main part. The roof box is then replaced by a mansard roof while the main part is processed by a facetization. After that, the facade design starts: first vertically split into floors, and then further split into a sequence of walls and windows at the floor level. Some floors may receive a running balcony. Going further, the windows are intruded, and decorated with an ornament and a small balcony. The complete set of slightly simplified rules is given in Table 2.5. Fig.2.15 shows the construction step of a Haussmannian building, where the basic shapes are colored boxes. Ultimately examples of high detailed generations are given in Fig.2.16. Walls are mapped with textures and hand-made 3D models are plugged for ornaments, balconies, doors and windows.



Figure 2.16: Procedural Haussmannian Buildings. 3D models are inserted at the terminal level.

2.4 2D Binary Split Grammars

So far, we have defined shape grammars and discussed their use in 3D building modeling. Since the goal of this work is to perform single-view image-based procedural modeling, we need to define grammars that are capable of describing the semantic structures of rectified images of facades. Thus, we define a new subclass of context-free shape grammars that only process 2D axis-aligned rectangular shapes and are particularly well suited to model facades. Moreover, following the same spirit as the *Chomsky Normal Form* for formal grammars (see section 2.1), we define here binary grammars which parse trees are binary (see Fig.2.17), and we call these grammars *2D Binary Split Grammars* (BSG).

2.4.1 Definitions

Restricting the authorized primitives to axis-aligned rectangles allows us to simplify the previous definition of shape grammars. Let's review the basic concepts defined in section 2.2.

Basic Shapes → **Symbols** Since we are only dealing with rectangles, we can actually replace the term basic shapes by *symbols*. We simply represent the basic shape by a name, and refer to them as *symbols*.

Scopes → **Rectangles** They position the shape in the 2D space. Since, we only allows axis aligned rectangles, a scope is now defined by a quadruplet (x, y, w, h) , defining the position of the lower left corner, and the width and height of the rectangle. We will now refer to scopes as *rectangles*.

Atomic Shapes Remember that an atomic shape is the association of a basic shape and a scope, and represent the instantiation of a basic shape. Now in the 2D case, we represent the atomic shapes as $\text{name}(x, y, w, h)$, since it contains all the relevant information.

1.	floorWall	\rightarrow_X	wall(x).floorWin + wall	$x \in [50, 120]$
2.	floorWin	\rightarrow_X	window(x).floorWall + wall	$x \in [50, 120]$

Table 2.1: The binary floor BSG. It has two rules, two non-terminal symbols and two terminal symbols. This grammar can handle any number of elements through the use of mutually recursive rules.

The language $L(G)$ is the set of all possible derivations of the grammar. Every shape in $L(G)$ is only made of terminal atomic shapes that are all axis aligned semantic rectangles, completely filling the rectangle of the axiom. If G is a BSG, where the axiom is the dimension of an image \mathcal{I} , then a shape $C \in L(G)$ is a 2D image of symbols that can be interpreted as a semantic segmentation of \mathcal{I} , since:

$$\forall x \in \mathcal{I}, \quad C(x) \in \mathcal{T} \quad (2.10)$$

A natural question one can ask, is the expressive power of BSGs. In the next section, we will see that binary splits are obviously as powerful as any kind of splits, and that this formulation is particularly well suited to deal with an unknown number of elements in a very compact way.

2.4.2 Expressive Power of BSGs

Proposition 1. *Every n -ary split can be represented by $n - 1$ binary splits, for all $n \geq 2$.*

This proposition is straightforward to prove by induction. Thus, the binary splits are as powerful as n -ary splits, and using them in a BSG does not limit the expressive power of such a grammar. Furthermore, binary splits present several advantages over n -ary ones, such as the natural way they handle recursion, and the compactness of a grammar that would contain only binary splits, over one containing n -ary splits. We are now going to explain these two properties which are a key to compact and powerful modeling of facades.

2.4.3 BSG for Compact Accurate Modeling

Remember that we want to model facade with different geometric parameters. The great power of binary split lies in the use of recursive rules. If a floor is made of a sequence of walls and windows, starting by a wall, we can model it with only two kinds of rules:

This grammar is made of two split rules, and a mutual recursion. A floorWall starts with a wall, and a floorWin starts with a window. Therefore, a floorWall can be seen as a wall followed by a floorWin and vice-versa. The alternation of walls and windows stops when the LHS is too small to be split. As a consequence, this formulation makes no exact assumption over the topology of the floor, that is to say the number of walls and windows. The illustration of this

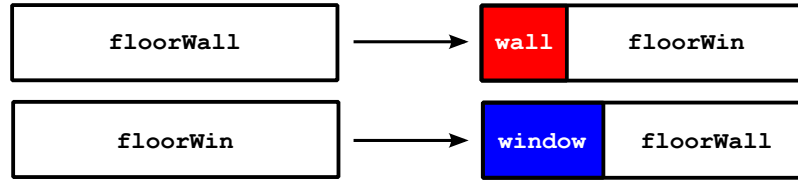


Figure 2.18: Modeling a floor using mutually recursive binary splits.

combination of rules is given in Fig.2.18, where the windows are represented in blue and the walls in red.

What would be the difference between using a single n -ary split or two mutually recursive binary split? First, as we mentioned earlier, two recursive binary splits allow us to get rid of an assumption on the number of elements in the floor. With a single split (without recursion) we can expect at most n elements if the split is a n -ary split.

The second advantage is the compactness of the grammar. For example, let's suppose that we may want to model a floor made of a sequence of windows which size varies between 30 and 50 pixels and walls which size ranges from 10 to 70 pixels. In the rest of the chapter, we consider that each split rule is defined with fixed parameters.

As an example, let's consider a n -ary split rule that has 5 windows and 5 walls. Since we have 20 possible lengths for a window and 60 for a wall, the number of split rules is $20^5 60^5 \approx 2.15 \cdot 10^{15}$. Using two binary split rules the number of rules in the grammar shrinks to $20 + 60 = 80$ rules. In the end, we will be able to model the same splits (provided that the number of elements remains smaller than 10) by applying the binary rules many times rather than by applying once a n -ary split rule. Needless to say that keeping in memory 80 rules costs nothing, while $2 \cdot 10^{15}$ is simply impossible. As a consequence, BSGs are both more powerful and more compact. They can express a greater number of buildings with a smaller number of rules by keeping a single parameter per rule.

Thus, BSGs seem to be very well suited to build shape grammars that can describe facades with an unknown number of elements in a very compact representation. Before explaining how to parse BSGs in chapter 4, let's first give some examples of grammars modeling different kinds of facades.

2.4.4 Some Examples of BSGs

We give here three examples of facade layouts that we can model with a BSGs. Please, pay attention here that the grammar presented so far is on purpose context-free. Therefore there is no reason why the facade should be consistent. The windows on the different floors have no reason to be aligned while deriving randomly the grammar. We will explain how this apparent drawback of context-free grammars turns out to be an advantage while properly optimizing them.

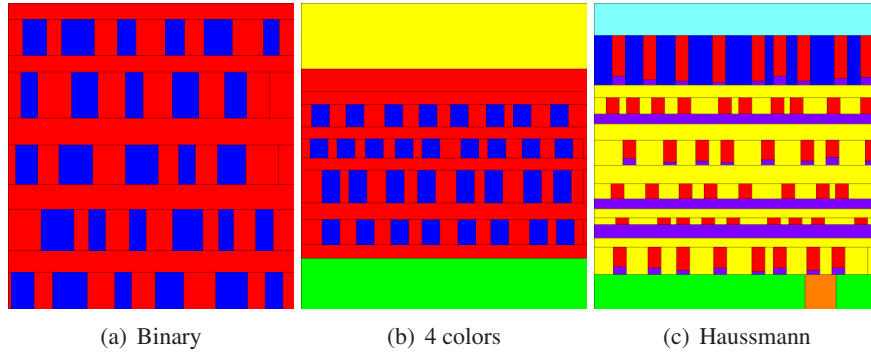


Figure 2.19: 3 randomly generated semantic segmentation using a BSG. Since a BSG is context free, alignment is not guaranteed. (a) Binary Segmentation grammar. (b) The 4-color grammar. (c) Haussmannian grammar.

1.	facade	→	fcdWa	\emptyset
2.	facade	→	fcdFl	\emptyset
3.	fcdWa	→ _Y	wall(y).fcdFl + wall	$y \in [50, 100]$
4.	fcdFl	→ _Y	flWa(y).fcdWa + wall	$y \in [100, 160]$
5.	flWa	→ _X	wall(x).flWi + wall	$x \in [50, 120]$
6.	flWi	→ _X	window(x).flWa + wall	$x \in [50, 120]$

Table 2.2: A binary grammar: wall/window.

Binary Segmentation

The idea here is to split a facade into a sequence of floors and walls and to split each floor into a sequence of walls and windows. Thus in the end, a segmentation only contains wall and window symbols. Here, we give the rules with generic parameters and the ranges of values these parameters can take. In the complete grammar each rule is duplicated as many times as necessary so that each one has a fixed and different parameters. This grammar has 6 different kinds of rules, and a total of 256 rules. We usually reduce a little bit the number of rules, by using only one parameter over 3. Therefore the number of rules is in practice 88. Please note that the size of the language of the grammar is much bigger. To give an order of magnitude of the cardinal of $L(G)$, consider that if we can apply 10 different rules at each node, and we are applying 60 rules to obtain a complete segmentation, then we can generate 10^{60} different symbolic buildings only for this topology. Knowing that the number of rules to apply may vary, and that we usually have more choices than 10 rules per node, then the number of possible segmentation is clearly untracktable.

An example of symbolic facade generated by this grammar is given in Fig.2.19(a).

1.	building	\rightarrow_Y	shop(y).upFcd	$y \in [150, 300]$
2.	upFcd	\rightarrow_Y	facade.roof(y)	$y \in [100, 250]$
3.	facade	\rightarrow	fcdWa	\emptyset
4.	facade	\rightarrow	fcdFl	\emptyset
5.	fcdWa	\rightarrow_Y	wall(y).fcdFl + wall	$y \in [50, 100]$
6.	fcdFl	\rightarrow_Y	flWa(y).fcdWa + wall	$y \in [100, 160]$
7.	flWa	\rightarrow_X	wall(x).flWi + wall	$x \in [50, 120]$
8.	flWi	\rightarrow_X	window(x).flWa + wall	$x \in [50, 120]$

Table 2.3: The 4-colors grammar: wall/window/roof/shop.

4-colors Facades

Sometimes a mere binary segmentation is not enough. Therefore we augment the previous grammar with two terminal symbols: shop and roof. This extended grammar has 8 different kinds of rules, and a total of 150 rules if we consider one parameter over 5 for the two first kinds of rules. See table 2.3 for details of the rules and Fig.2.19(b) for an example of a random generation of facade.

Hausmannian Facades

Ultimately, we propose a more complex facade for Hausmannian building modeling. This grammar contains 7 different terminal symbols. For this grammar, we used other operators, such as an operator to set a parameter for a whole building (the width of the windows), and an operator that acts as a differentiation tag operator, in order to consider the roof loft as a floor, and replace the wall by a roof symbol. Moreover, it also allows a running balcony to optionally appear on any floor. This grammar is slightly more complex than the previous ones, but globally works with the same mechanisms. It has 351 rules in total. See Fig.2.19(c) for an example of an image sentence generated by this grammar.

2.5 Conclusion

Procedural modeling as defined in this chapter is a powerful framework to represent complex geometries such as buildings or other types of architectural structures. It succeeds in catching both the uniformity of an architectural style and the diversity of its instances. These procedural models have three main advantages:

Compactness Grammars are made of a few number of rules. However, their combinations can generate an almost infinite number of instances.

Expressiveness A single grammar catches a great diversity of buildings, in terms of geometry and topology. Using the same grammar, one can generate a building with 2 floors or a building with 100 floors, without any more difficulty. From a generation point of view, this is very good news. Designing a shape grammar may be an investment, but then a designer can generate as many instances as he wants, at almost no extra cost. However, from an optimization point of view, this huge expressiveness power is not necessarily good news. How to optimize a model for which we do not even know the number of degrees of freedom?

Stochasticity Another interesting aspect is the possibility to have stochasticity, both in the rule parameter and in the derivation process. This property makes it possible to sample randomly shapes in the language of a grammar, and will be intensively used for optimization.

In spite of these very nice properties of shape grammars, the generated building shapes are somewhat disappointing. Even with realistic textures, the buildings do not look realistic at all: they look “procedural”. This feeling is even more important while generating a large-scale environment. The regularity of shapes and textures are even more visible. This is almost a complete failure! While the cost of procedural modeling becomes interesting only for large scale environment, the visual quality of the generated scene decreases with the number of buildings. Therefore, designing a grammar for an architectural style seems to be a waste of time.

However a simple experiment might save procedural modeling. Put the photograph of a building as a texture for the corresponding procedural model, mixing 3D models for fine elements and real textures. The result is indeed very impressive and very realistic. It seems that the magic of procedural modeling is bound to real data. This raises a question: how to match the texture and the procedural model? If the texture of a wall falls on a 3D model of a window, then the final model looks very bad. The only way to make it good is to manage to match the element of the image with the elements of the procedural model.

This problem can be seen from the two sides. The first one is to find the building that fits the input texture, and the second one is to synthesize a texture that fits the building. Chapter 3 brings a bottom-up approach to the first problem and chapter 4 and chapter 5 present the theory and application of a generic top-down approach. The second question of synthesizing textures is left as future work.

1	axiom	→	Scale _Z (volume:H)
2	volume	→	subvolume + Translate _Z (Scale _{X,Y} (roof:1.05):1r)
3	subvolume	→	Split _Z (stairs:2a, temple:1r)
4	temple	→	columns + Shrink(naos:3a)
5	roof	→	Hip(hroof)
6	stairs	→	Split _Z (step:0.6a, Shrink(stairs:0.4a):1r)
7	columns	→	Facetize(side:street, front:neighbor)
8	side	→	Split _Y (peristyle:1r, entablature:4a)
9	entablature	→	Split _Y (architrave:2a, cornice:0.3a, frieze:1r)
10	frieze	→	Mirror _X (half frieze:1r, triglyphe:0.8a)
11	half frieze	→	Repeat _X (trig box:1.6a)
12	trig	→	Split _X (triglyphe:0.8a, stone:1r)
13	architrave	→	Repeat _X (stone:4a)
14	peristyle	→	Mirror _X (peri:1r, column:1.6a)
15	peri	→	Repeat _X (Split _X (column:1.6a, ∅:1r))
16	front	→	Mirror _X (Split(chunk:2.4a, side:1r))
17	chunk	→	Scale _Y (Split _Y (stone:2a, cornice:0.3a, trig:1r):4a)
18	column	→	Extrude(Translate _Z (col:-1r):1.6a)
19	col	→	Split _Y (shaft:1r, capital:1a)
20	capital	→	Scale _{X,Z} (abacus:1.1r)
21	shaft	→	Repeat _Y (section:12c)
22	section	→	Scale _{X,Z} (piece:entasis(z))
23	triglyphe	→	Split _X (glyphe:0.2a, Repeat _X (hole:0.2a, glyphe:0.2a):1r)
24	glyphe	→	Extrude(glyphe3D:0.3a)
25	hroof	→	Facetize(Shrink(Extrude(sima:0.2a), -0.2a):neighbor, tile:street)
26	tile	→	tile + Scale _Y (Extrude(antefixe:0.5a):0.3a)
27	antefixe	→	Repeat _X (Split(ornament:0.8a, ∅:1r):1.5a)
28	cornice	→	Extrude(cornice3D:0.2a)

Table 2.4: Doric Temple Shape Grammar.

1	axiom	→	Scale _Z (volume:h)
2	volume	→	subvolume + Tranlate _Z (Scale _Z (roof:h _{roof}):1r)
3	roof	→	Mansard(loft,top,α _{loft} ,h _{loft})
4	subvolume	→	Facetize(Ctag(facade:1c):street,fake:neighbor)
5	facade	→	Split _Y (GF:h _{GF} , Split _Y (sFloor:h _{FL} ,inter:h _{inter} ,sFloor:h _{FL} ,...):1r, cornice:0.2a) + Translate _Y (Scale _Y (Dtag(sFloor:1c):h _{loft}):1r) + Translate _Y (Scale _Y (Dtag(chimneys:2c):h _{chimney}):h+h _{loft})
7	sFloor	→	Dtag(Ctag(cFloor:sFloor.z):tag)
8	chimneys	→	Extrude(Translate(chimns3D:2a):-1r)
9	chimns3D	→	Repeat _X (Split(∅:1r,chimney:1a,∅:1r):4a)
10	inter	→	Split _Y (wall:1r,cornice:0.2a)
11	cornice	→	Extrude(cornice3D:0.2a)
12	cFloor	→	Scale _Y (Extrude(balcony:0.8a):h _{balcony}) + floor
13	cFloor	→	floor
14	floor	→	Split _X (wall:w _{wa} ,Ctag(tile:w _{tl} :tag),wall:w _{wa} ,...)
15	tile	→	Switch(rTile:dtag==1,fTile:dtag<0)
16	fTile	→	Split _Y (sWin:1r,ornament:h _{or})
17	fTile	→	Split _Y (sWin:1r,wall:h _{or})
18	sWin	→	Scale _Y (Extrude(balcony):h _{balcony}) + Extrude(window:-0.4a)
19	ornament	→	Extrude(ornament3D:0.3a)
20	GF	→	Split _X (shop:w _{shop} ,door:w _{door} ,shop:w _{shop})
21	shop	→	Extrude(shop3D:d _{shop})
22	rTile	→	Extrude(Translate _Z (rWin:-1r):h _{loft} /tan(α _{loft}))
23	wall	→	∅ if dtag==1

Table 2.5: Haussman Buildings Shape Grammar.

Chapter 3

Bottom-Up Parsing

This chapter introduces a method to perform bottom-up facade parsing. The goal of this chapter is to identify in a single rectified image of a facade the positions of the windows, under an alignment assumption. To perform this task, we rely on grouping and analyzing extracted feature points. The main contribution of this work is to relax the regularity hypothesis usually assumed in facade parsing. On the one hand it allows us to analyze irregular building facades in an unsupervised way. On the other hand such a move has a cost; the proposed method is bound to the use of heuristics. After reviewing the state-of-the-art in repetitive structure detection and facade analysis in section 3.1, we will explain how to extract relevant information in section 3.2, how to interpret it in section 3.3 and how to gather it in a consistent graphical model in section 3.4. Section 3.5 presents the results of the proposed approach on different building facades and section 3.6 concludes the chapter and discusses the relevance of bottom-up parsing and its limitations.

3.1 State of the Art

3.1.1 Introduction

Facade image analysis has recently become a very active field in the Computer Vision community. This increase of interest can certainly be linked to the release of on-line databases of street images such as Google Street View or Microsoft Bing Maps, which present realistic ways to traverse geo-referenced images. These applications are stitching images together to create an illusion, but they hardly analyze the image contents. Such databases of images call for intensive automatic processing. Having a reliable tool to parse these images would open the door to large-scale city reconstruction, bridging the gap from 2D stitched images to realistic 3D environments with numerous potential commercial applications such as realistic video games. As a consequence, several methods have been proposed to tackle our problem of single-view facade analysis and window detections in urban environment in the last few years.

An impressive amount of work has been proposed to tackle the problem of facade interpretation as a top-down parsing problem, related to procedural modeling. These works will be reviewed into

details in chapter 4. There are mainly two drawbacks of such approaches: first they need a complex model (usually a shape grammar) which implies a procedural engine to generate shapes and the time-consuming task of designing the grammar itself. Then, these methods are computationally heavy and usually have to rely on strong image cues in order to speed-up their convergence rates. Thus, they raise the legitimate question of the need of complex models. It sometimes seems that the image cues or the heuristic involved are more important than the complex models themselves, questioning their relevance and their necessity.

There are basically two hypotheses while dealing with facade images in a bottom-up way. The first one is close to procedural modeling ideas. Since the facade can be represented by a grammar, some elements such as the windows should be “copy-pasted” regularly on the image. Therefore, some bottom-up methods are incrementally grouping similar image regions to build the facade. For that matter, these methods study repetitions, similarities and symmetries. The second hypothesis is more low-level, it assumes that the windows of the facade should have a more complex appearance than the wall, leading to image regions with stronger gradients. Thus, some methods are focusing mainly on studying the gradient signals on the image. Let’s review both kinds of methods.

3.1.2 Repetitions and Lattices

A first class of methods focuses on regular repetitions in generic images. They are studying Near Regular Textures (NRTs). Such models can be represented by 2D lattices. In other words there exist two vectors t_1 and t_2 that generate the lattice. Each point is obtained by a integer combination of these two generator vectors. In one of the pioneer works in repetition detection [Leung 1996], the authors propose an algorithm in four steps: interest point detection, matching, clustering and lattice growing. In their work for instance, they detect corners using a Harris-like detector, and match the different points using a sum of squared differences (SSD) criterion. This key paper also turns the repetition detection problem into a tracking problem. As often, this change of viewpoint is very fruitful, and most of the following works were somehow extensions of this one.

Inspired by this work, many other groups have applied the same generic scheme for repetition detection. For example, [Schaffalitzky 1999] proposes the same generic scheme, using again Harris corner detector coupled with normalized cross-correlation (NCC) on small patches surrounding the interest points. Turina et al. [Turina 2001] make use of another points detector and descriptor called affine invariant neighborhood. Coupled with Mahalanobis distance and Hough transform, they obtain very good repetition detections. NRTs were a popular topic in the middle of the 2000’s. Liu et al. [Liu 2004, Liu 2005] use the crystallography groups to characterize automatically near regular textures, while the authors of [Hays 2006] still improve the lattice detection with texels, following the tracking ideas originally proposed by [Leung 1996].

Little by little, people are applying repetition detection to urban environments, trying to discover lattices in urban scenes. New features and new optimization tools are also introduced. [Wenzel 2008] also detects repetitions, and follows ideas that were first developed by the 3D community [Mitra 2008, Pauly 2008]. SIFT points are detected on a rectified facade, and each pair of similar SIFT casts a vote in a transformation space that reveals the underlying lattice.

The same kinds of idea were also exploited in [Musialski 2009]. Lattices again are detected directly on perspective images by [Schindler 2008], using also SIFT features and RANSAC. The estimated lattices are used to recover 3D information on the scene. An interesting work by Korah et al. [Korah 2006] detects window lattices, based on rectangle detections and a MRF formulation, solved through a MCMC method. Once discovered, the window lattice is used to perform texture enhancement, facade segmentation and automatic rectification. The most recent and probably most advanced work on lattice detection was proposed by Park et al. [Park 2008, Park 2010]. The authors extract several types of features (SURF [Bay 2006], KLT [Lucas 1981], MSER [Matas 2004]) and take advantage of their complementarity to group them efficiently and discover very complete lattices. Note that another work on repetitions was used to segment individual facades [Wendel 2010] in images that contain several ones. Starting from the same observation that the image might contain several buildings or several modes of repetitions, the authors of [Yakubenko 2010] detect multiple regular grids in rectified images.

The great majority of the aforementioned approaches shares some common properties: they are based on features detection and matching and assume strong regularities. Thus, feature matching was replaced by entire region comparison with mutual information in [Müller 2007]. Furthermore, in [Koutsourakis 2009b] we compare regions using SSD and encode structural constraints in a Markov chain to impose consistency to the facade layout. Please note that this last work may be among the only ones not to assume regular repetitions.

3.1.3 Axial Symmetries

Axial symmetries are another kind of clue that is important to take into account while dealing with detection. The literature on symmetries is very broad. A first and very complete approach was proposed by [Kiryati 1998] that does not rely on interest point extraction at all. A gray scale image is seen as a function of two variables and the authors detail how to characterize and detect local symmetries in such a function. The optimization is quite heavy and based on genetic algorithms. However, this work is quite uncommon in the literature. Usually the symmetry detection is based on matching features, as it is the case with repetition detection. This idea was followed by [Scognamillo 2003] and improved by [Loy 2006] by introducing efficient SIFT features and their mirrored versions. Similar ideas were also applied to 3D models, in which the interest points are usually better described than in images. Therefore, analyses of 3D shapes often lead to very impressive results. The authors of [Mitra 2006] detect axial symmetries in various 3D models.

Symmetries and repetitions were actually already coupled in [Korah 2008] for facade analysis and more recently, [Wu 2010] proposed a systematic analysis of repetitions in urban scenes that also takes advantage of symmetries. A main difference with other works is that they only consider repetitions along the horizontal axis. However, as in most of the current work dealing with facade analysis and repetition detection, they make a hypothesis of 3D regularity (the images are rectified automatically). This successful approach was then extended to single-view 3D reconstruction [Wu 2011].

3.1.4 Window Detection

As mentioned earlier, another way to approach facade analysis is to start from the very simple hypothesis that the windows of the facade should be easy to detect, either using the common heuristic that the strong gradients of the image correspond to windows regions or by applying a supervised learning on windows.

[Lee 2004] was the first work to introduce gradient profiles as an efficient way to detect windows in rectified images. Many works have been inspired by this one [Hernández 2009] and also [Burochin 2009, Liu 2010]. Besides, [Recky 2010] uses a K-Means on CIE-lab color space as a heuristic to detect windows in images. Eventually, a supervised learning approach was envisioned by [Ali 2007] that makes use of AdaBoost classifiers or by [Reznik 2007] who introduce Implicit Shape Models to represent windows.

However, these approaches are certainly not as elegant as the ones based on symmetry and repetition detections. As a consequence, we opted for a mixed approach, based on repetition detection and studying the repetition signal. Furthermore, we had to relax the regularity constraint that is clearly the common denominator to all the aforementioned approaches. Let's discuss the reason why this relaxation is necessary.

3.1.5 Relaxing the Regularity Constraint

The aforementioned methods are performing very well on regular facade images, such as skyscrapers; because they usually assume that the repetitive pattern (the window) is repeated according a regular transformation (a 2D lattice). However, in many buildings, such as the Parisian ones, the repetitions are not regular. First, the pattern itself does not have a uniform size: the windows on the second floor may be higher than the windows on the third one. Then, the distance between the windows may vary as well. Indeed the facade layout cannot be represented by a 2D lattice. There exists no couple of vectors \mathbf{t}_1 and \mathbf{t}_2 that can explain the positions of the windows on the facade.

As a consequence, in order to discover the windows layout of the facade there is no choice but to relax the hard constraint of a 2D lattice. However, removing completely any structural assumption is bound to fail: the complexity of the window pattern in a facade may lead to false detections or missed targets, and the frequent presence of large occlusions in such images requires a structural assumption. The most generic and sensible structural assumption is *alignment*. Although it could sometimes be violated in modern architectures, such an assumption is respected in the large majority of facade layouts. We will consider this hypothesis in the remainder of the chapter. Going beyond this assumption require a more complex modeling of facades, and we propose in chapter 4 a more generic approach to facade parsing that encompasses all the possible cases through the direct use of shape grammars.

Thus, in this chapter the shape grammars are not used directly as input of the algorithm, but rather as an output. We assume that most of the building facades can be expressed by very simple 2D shape grammars that generate irregular grid layouts. Since a 2D shape grammar repeats the same symbols at different location of the 2D plane, it is quite natural to make the assumption that

these symbols correspond to image regions that look alike on the image. It raises the questions of grouping these similar regions (see section 3.2), analyzing these consistent groups (see section 3.3), and enforcing the structural constraint in a single graphical model (see section 3.4).

3.2 Detection of Similar Features

A repeated pattern in the image should induce similar interest points and regions. In this section, we study how to detect, describe and group them.

3.2.1 Interest Point Detection

The low-level vision literature contains many kinds of primitive detectors: points (corners, blobs, etc.), edges or even fancier types of primitives such as rectangles. A recent trend in Computer Vision focuses on the definition and use of super pixels [Mori 2004, Moore 2008, Levinshtein 2009]. Even if points are supposed to be less robust than more complex primitives, they remain easy to detect and easy to describe, which explains their popularity. Moreover, considering that we are using rectified images of facades, usually taken from an original almost frontal view, the detection of interest point is quite robust, except in presence of occlusions. For all those reasons, we first focus on detecting interest points in the image.

Many interest point detectors have been released since the early days of Computer Vision. They are classically based on signal processing principles: the interest points are maxima of some functions computed on the image signal. For instance, Harris and Stephen [Harris 1988] proposed a corner detector based on the eigenvalues of the autocorrelation matrix. [Mikolajczyk 2004] later made it affine invariant. Probably the most important work [Lindeberg 1998] introduced the idea of the scale space to detect interest points associated to a given scale. The same idea was then used for what is today the most popular interest point detector: The Scale Invariant Feature Transform (SIFT) [Lowe 1999, Lowe 2004]. It combines an interest point detector based on the maxima of the scale space with an efficient descriptor that relies on histogram of gradients. Besides, good detection and matching results in many applications as well as an efficient available implementation by Andrea Vedaldi [Vedaldi 2008] made it very popular. More recently, [Bay 2006] released another similar interest point detector and descriptor called SURF. Other ideas were also proposed. For instance, Matas et al. proposed a blob detector based on maximal stable regions [Matas 2004], while salient regions have been introduced by [Kadir 2004] making use of information theory.

Recently, low-level vision started taking advantage of Machine Learning with outstanding performances. The idea behind it is to replace a heuristic function to be maximized (scale space, eigenvalues of the autocorrelation matrix, etc.) by a classification function to be learned from examples. This trend lead to improvements in boundary detection [Martin 2004, Kokkinos 2010] and in corner detection such as FAST [Rosten 2009]. In this last work, the authors propose a learning-based corner detector that actually performs faster and better than the previous state-of-the-art.

Seeking for windows, a corner detector seems to be more appropriate than a blob detector, which would provide interest points in the middle of flat regions. Instead, detecting interest points

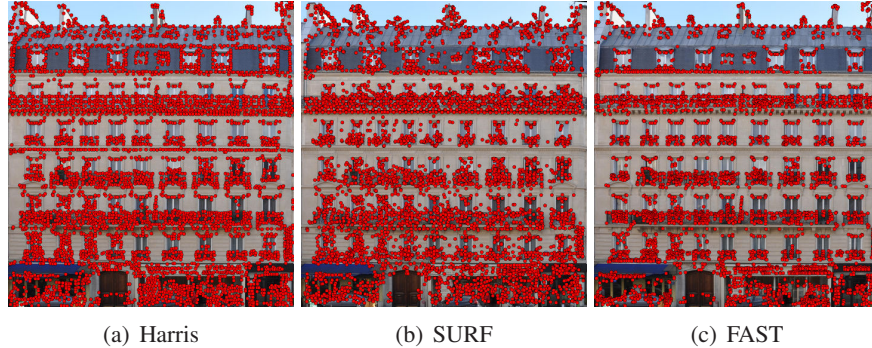


Figure 3.1: Different interest point detectors applied on a facade image.

mainly on the repetitive pattern helps to discover the pattern layout. In Fig.3.1, we show the responses of different interest point detectors on a typical facade image. Qualitatively, the points obtained using FAST are better detecting the windows than the other ones that tend to detect interest points everywhere on the image. As a consequence, the FAST detector will be considered in the proposed approach.

After the interest point detection, we end up with set \mathcal{K} of N keypoints:

$$\mathcal{K} = \{p_i = (x_i, y_i)\}_{1 \leq i \leq N} \quad (3.1)$$

3.2.2 Interest Point Description

Once some interest points have been detected on the image, the next step aims at quantifying how much two points look alike. This implies associating a feature vector to each point and comparing the two corresponding feature vectors. Let's first study the problem of interest point description.

In the literature, many descriptors have been proposed. A descriptor should be both discriminative and robust to changes in viewing and illumination conditions. Usually, this goal is formulated as affine invariance. All the proposed descriptors aim at capturing the local context of a given region on the image. For example, medical imaging uses Gabor filters bank responses a great deal [Zhan 2003]. Although Gabor filters are usually good texture descriptors, they are performing a time-frequency analysis of a signal which is sometimes slow and not appropriate in Computer Vision. Indeed, the local appearance of images may be too complex to be studied through the prism of a coarse time-frequency analysis. Depending on the level of invariance needed (view-point, scale, orientation, illumination, etc.) the choice of descriptor may vary. In some applications, mere patches around the interest points are used as descriptors. For example Lepetit and Fua [Lepetit 2006] use patches to train classifiers for object detection. Usually though, people try to use more invariant features.

The Scale Invariant Feature Transform [Lowe 1999] is based on histogram of gradients. An

oriented grid of 4 by 4 cells of same sizes collect the histogram of gradients along 8 different directions. Therefore the SIFT descriptor is 128-dimensional. The scale and rotation invariance come from the size of the grid and its orientation. Usually, the SIFT description is preceded by a detection step that finds the maxima of the scale-space, selecting image points x, y at scale σ and with orientation θ (the orientation of the gradient). However, the detection step is not necessary, and the SIFT descriptor can be applied on any point of the image, at any scale and any orientation.

A very interesting study by Mikolajczyk and Schmid [Mikolajczyk 2004] showed that SIFT-based descriptors outperform their competitors, especially GLOH (gradient location-orientation histograms), which is basically an extension of SIFT, with a different grid and using PCA as a way to decrease the dimension of the feature vectors. Later, SURF descriptors [Bay 2006] turn out to be a good alternative to SIFT. However, a common criticism about all the aforementioned descriptors is that they are *ad hoc*. They are all based on sensible principles, but they turn out to make some debatable choices, such as the discretization level, the number of bins or orientations chosen. For that matter, another study by Winder and Brown [Winder 2007] rethinks the description process more globally, and proposes to learn the parameters rather than to hard code them. Not surprisingly, this more elaborated approach that indeed embed the SIFT descriptor manages to outperform SIFT. Other SIFT-based approaches also extends the SIFT descriptor to make it affine invariant. For instance, Morel's ASIFT [Morel 2009] has proved to be more robust than regular SIFT.

Despite the existence of better descriptors than SIFT, Lowe's descriptor remains the gold standard in Computer Vision tasks. Indeed, SIFT was the first descriptor providing such good performances in matching, and a very good implementation has been released [Vedaldi 2008]. One of the main drawback of SIFT is indeed its complexity. For that reason, GPU implementations were first considered as a good alternative [Sinha 2006]. More recent efforts focused on descriptors that are faster to compute and faster to match as well. Kokkinos and Yuille [Kokkinos 2008] introduced SID, a descriptor that does not need the very time-consuming step of scale-space creation. DAISY [Tola 2010] is another descriptor based on SIFT ideas, using circular cells of increasing sizes. The farther from the point, the bigger is the region to be described. Then, the heavy computation of SIFT is here replaced by the use of convolutions, leading to a speed of 40 in practice with better matching performance than SIFT. Eventually, a last descriptor called LDAHash has been proposed by Strecha et al. [Strecha 2010]. It aims at reducing the size of the descriptors in order to speed-up the matching phase. The authors manage to turn them into short sequences of Boolean values. The Hamming distance between them can be very efficiently computed by bit-wise operation. This allows them to obtain very fast matching, even with a large number of feature vectors.

Although these recent works in feature description outperform the SIFT descriptors, we made the choice of keeping SIFT features for convenience reason. Indeed, they still perform very well and present the great advantage of having good available implementations used by the whole Computer Vision community.

After detecting with FAST some corner points, we use a SIFT descriptor with the same scale and the same orientation for all the corners. The reason for that choice is that we are processing rectified images of facades in which we are looking for translated patterns. Therefore, neither the orientation nor the scale of these interest points should vary. Orientation is fixed to 0 and scale is



Figure 3.2: Sift descriptors. The orientation is represented by the line inside the circles and the scale is represented by the radius of the scale.

fixed a value that can either come from a SIFT detector, or given by hand. In practice we used a fixed scale of 3. A multi-scale approach [Musialski 2009] could also be a good alternative to get rid of the choice of the uniform scale. Fig.3.2 shows a representation of the SIFT scales, orientation and position on the an image.

After the description step, we end up with a set \mathcal{F} of feature vectors that are points $p_i \in \mathcal{K}$ associated with descriptors $d_i \in \mathbb{R}^{128}$:

$$\mathcal{F} = \{f_i = (p_i, d_i) \in \mathcal{K} \times \mathbb{R}^{128}\}_{1 \leq i \leq N}. \quad (3.2)$$

3.2.3 Clustering in the Features Space

Remember that so far, we have extracted and described interest points on the facade. N feature vectors f_1, \dots, f_N represent either corners on the repetitive pattern, or outliers. In a bottom-up approach, the goal is to group together the similar points and discard the outliers so as to reveal the facade layout, which is assumed to be an irregular grid of repeated features. In all likelihood this grouping should correspond to a clustering of the feature vectors. To this end, one needs to choose an appropriate distance between feature vectors.

Choice of a Distance between Feature Points

The SIFT descriptor is basically a histogram. Even though many distances may be chosen between histograms, such as Kullback-Leibler divergence, Bhattacharyya distance or Earth mover's distance, the simple Euclidean (L_2) or L_1 distance are usually used together with SIFT features. We choose the L_1 distance.

Clustering Methods

Having chosen a distance, one needs to choose a clustering method. The most standard clustering method is definitely K-Means. This approach is very efficient when the number K of clusters is known, since it is an input of the algorithm. In our case, we do not know the number of clusters beforehand. In such a case, other methods can be envisioned. Mean Shift [Comaniciu 2002] is a very efficient algorithm. The idea is to find the attraction basins of a distribution, by shifting a search window towards the mean of its distribution until it converges towards a mode. Each mode corresponds to a cluster and all the samples in the attraction basin of a mode are associated to it. Mean Shift was successfully applied in many applications. However, the control on the algorithm is somewhat difficult to achieve, since the only degree of freedom is the size of the search window.

Another clustering algorithm proposed by Nikos Komodakis and called clustering via LP-stabilities [Komodakis 2009] offers a more intuitive control. This algorithm is based on linear programming. It aims at clustering a set of vectors $Y_i \in \mathbb{R}^d$ by minimizing the following energy:

$$E(K, Y_{j_1}, \dots, Y_{j_K}) = \sum_{k=1}^K c(Y_{j_k}) + \sum_{i=1}^N \min_k \|Y_i - Y_{j_k}\| \quad (3.3)$$

where $c(Y)$ is the cost of Y to be the centroid of a cluster. Usually, this cost is the same for all the vectors, but it could as well be different. The first observation is that the number of cluster K is an output of the formulation. A second observation is that, unlike K-Means or Mean-Shift, the centroids of the clusters are necessarily part of the data set.

By solving this equation, we obtain the optimal number of clusters K^* and the centroids of each cluster such that they minimize the distances between each vector and the centroid of the cluster it belongs too. The use of the minimum in the sum represents the assignment of each vector Y to a cluster i , which centroid is Y_{j_i} . If the number of clusters K and the centroids Y_{j_1}, \dots, Y_{j_K} are known, then each feature vector is associated to the cluster of the closest centroid.

In the proposed method, we use a fixed cost c for all the vectors. No vector is better suited to be the centroid of a cluster than any other one. The clustering energy has now a single external parameter c :

$$E(K, Y_{j_1}, \dots, Y_{j_K}) = Kc + \sum_{i=1}^N \min_k \|Y_i - Y_{j_k}\| \quad (3.4)$$

It is interesting to observe that c constitutes an intuitive control over the number of clusters. If a vector Y is at a distance greater than c to the closest centroid, then it costs less to create a new cluster centered on Y than to assign Y to an already existing cluster. There are two extreme cases. First, if c is 0, then the trivial solution is the one where each vector is its own cluster. The optimal solution is then made of N clusters. The second extreme case is when c is infinite, or at least very big. Then, creating a new cluster costs a lot, and therefore all the vectors will be assigned to the same cluster. The optimal solution is made of 1 cluster. The number of clusters K is a decreasing function of c .

Furthermore this clustering enables us to control the geometry of the clusters themselves. In any cluster, the distance between any vector and the centroid is always smaller than c . The triangular inequality shows that in any cluster, two vectors have a distance smaller than $2c$. Therefore, the choice of c directly impacts the geometry of the clusters.

Grid Clustering

Remember that we would like to group the feature vectors of \mathcal{F} , so that within each cluster, all the feature vectors look alike (similar descriptors) and the feature vectors are scattered on a grid (not necessarily regular).

The geometrical constraint cannot be achieved with points. If we have two points, it is not possible to decide whether or not they are lying on a common grid. Therefore, a theoretical solution is to perform a clustering on pairs of feature vectors. $Y = (f^1, f^2) \in \mathcal{F}^2$, with a specific cluster for outliers pairs. If the two feature vectors of a pair are not similar (the distance between their descriptors is bigger than a threshold), then the cost of assigning the pair to the outliers class is 0. Otherwise, the cost of assigning the pair of similar feature vectors to the outliers class is infinite. Then, the distance between two inlier pairs is the product of two terms: an appearance term that is the sum of the distance between the pairs of descriptors, and a geometry term that is 0 if the two pairs are aligned or orthogonal and infinite (or very big) otherwise. This clustering is based on both the geometry and the appearance of the descriptors.

However, in practice this method cannot be implemented because of time and memory limitations. The distance between two pairs involves 4 feature vectors. If the detection step has detected 10^3 interest points, then there are $(10^3)^4 = 10^{12}$ such pair-wise distances to be computed. Needless to say that keeping such big array of distances in memory is infeasible, and even the computation of such a huge number of distances would be very time consuming.

As a consequence, we do not explicit the exact clustering equation in that case, since any implementation is unrealistic. However, I wanted to point out that we could theoretically group the detected feature points into grid of similar points in a single shot. In practice, we replace this grid clustering by a clustering over the descriptors of the points, followed by a geometric filtering to discard outliers in each cluster (which are similar to the other elements of the cluster, but that are not respecting its intrinsic geometry).

3.2.4 Descriptor Clustering

Rather than clustering pairs of SIFT, we first perform a clustering directly on the descriptors of the feature points: $Y_i = d_i \in \mathbb{R}^{128}$. The clustering energy minimized by [Komodakis 2009] is therefore:

$$E(K, d_{j_1}, \dots, d_{j_K}) = Kc + \sum_{i=1}^N \min_k \|d_i - d_{j_k}\|_{L^1} \quad (3.5)$$

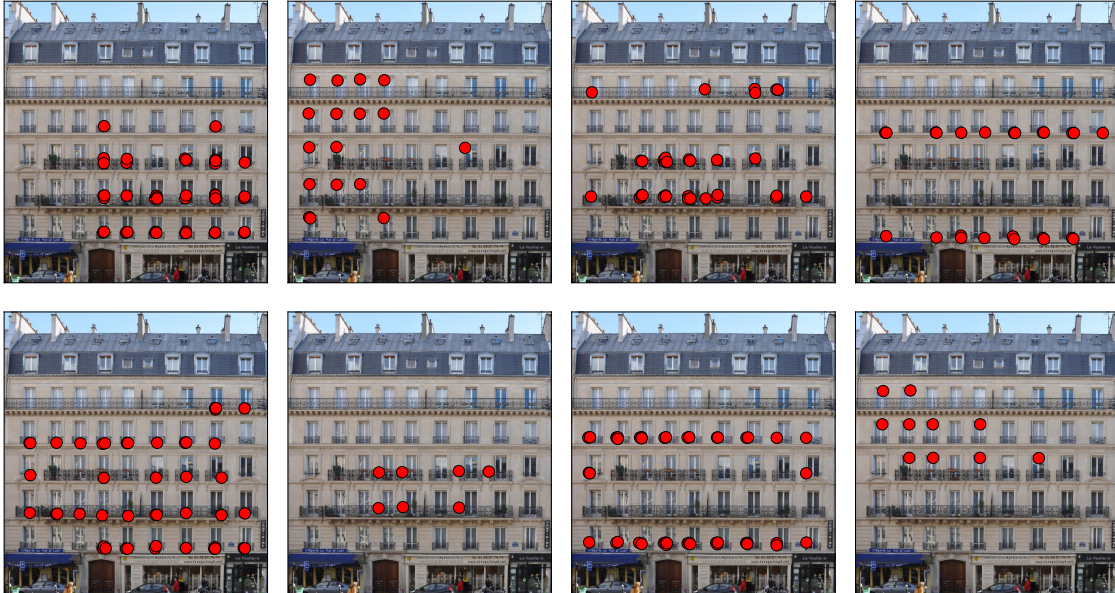


Figure 3.3: Some clusters obtained after clustering via LP-stabilities over the SIFT descriptors.

Minimizing the energy defined in equation 3.5 provides a set of K clusters of similar SIFT descriptors. In order to enforce the grid structure, we apply a post-processing step. In each cluster, we keep a feature point $f_0 = (p_0, d_0)$ at position $p_0 = (x_0, y_0)$ if there exist in the same cluster two other feature points at positions (u, y_0) and (x_0, v) with $u \neq y_0$ and $v \neq y_0$. This geometric filtering guarantees that each point of the cluster is lying on the same horizontal and the same vertical of two other points. Therefore these three points are three nodes of a common grid. All the feature vectors that do not satisfy this property are discarded. Fig.3.3 shows different clusters obtained after the geometric filtering step.

3.3 Cluster Analysis

After clustering the SIFT descriptors and applying a geometrical filtering of non-grid points, we obtain K consistent subsets of interest points (see Fig.3.3). They all represent visually similar points of the repeating pattern and they are all spatially organized according to a grid structure. However, it might happen that some clusters contain very meaningful information while others are not that informative or even represent pure outliers. Therefore we need to perform two main tasks in each cluster. First, we aim at extracting the geometric information relative to the pattern. Second, we would like to evaluate the confidence or score of the cluster itself. According to the shape grammar assumption, the horizontal pattern repetition is supposed to be independent from the

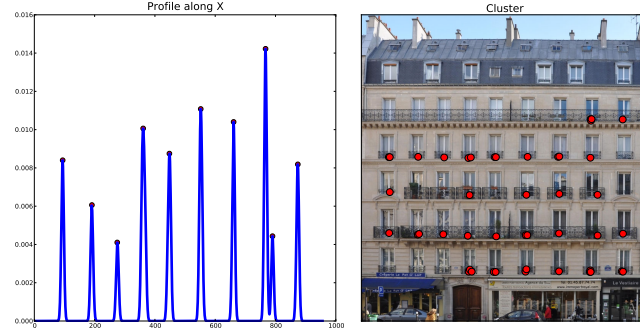


Figure 3.4: Example of profile along X and the corresponding extracted candidates.

vertical repetition. As a consequence, we tackle the two problems independently. In the remainder of this chapter, we will only consider the problem in the horizontal direction. The problem along the vertical one is solved similarly. There are mainly two random variables we can estimate in each cluster: the position X of the grid points and the size J of the repetitions between two consecutive patterns. We call it the “jump”.

3.3.1 Position Distribution

Let X be the random variable that represents the position of the pattern horizontally. Let N_k be the number of elements of cluster k , and $\{p_i^k = (x_i^k, y_i^k)\}_{1 \leq i \leq N_k}$ the elements of cluster k . Then the probability of having a pattern at position x is given by:

$$p(X = x|k) = \frac{1}{N_k} \sum_{i=1}^{N_k} \delta(x_i^k, x), \quad (3.6)$$

where δ is the Kronecker symbol. An example of such a profile is given in Fig.3.4. We can notice that the extracted profile is very sharp, and corresponds to actually possible location of the windows. However, we cannot know a priori if these positions correspond to the beginning or the end of the pattern.

3.3.2 Jump Distribution

Let J be the random variable that represents the distance between two *consecutive* patterns. We call it “jump”. Under equiprobability assumption, for each pair of positions x, x' with $x < x'$ in cluster k , we compute the probability that x and x' represent the same points on two consecutive patterns:

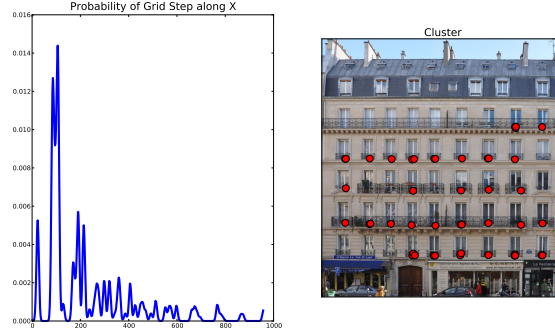


Figure 3.5: The jump probability from a window to the next one.

$$\begin{aligned}
 p(J = \gamma|k) &= \sum_x p(X = x + \gamma|X = x, k)p(X = x|k) \\
 &= \sum_x p(X = x + \gamma|k)p(X = x|k)
 \end{aligned} \tag{3.7}$$

under independence assumption.

Another heuristic possibility is to weight the probability $p(X = x + \gamma|X = x, k)$ by the integral of the distribution between x and $x + \gamma$:

$$p(J = \gamma|k) \propto \sum_x \frac{p(X = x + \gamma|k)}{\int_x^{x+\gamma} p(X = t) dt} p(X = x|k). \tag{3.8}$$

This means that the more elements are lying between x and $x + \gamma$, the less probable it is that the patterns at x and $x + \gamma$ are consecutive. Thus, the less probable it is that γ is a good jump between two *consecutive* patterns.

Furthermore, we actually add two fake positions while computing the jump distribution: an element at position 0 and one at position w (the width of the image). This trick allows us to catch all the transitions in the images from a point to the next one, included the one with the borders of the image domain. We now have an estimation of probability distribution $p(J = \gamma|k)$. Example of such jump distribution is given in figure 3.5.

3.3.3 Cluster Score

The goal of this step is to discard the clusters that do not represent well the repetitions of the pattern over the facade, by computing a cluster score. Indeed, some clusters may gather points that are not

clearly correlated or mere outliers. Hence, the score would ideally reflect the geometry of the cluster, its relative regularity and its mass. Therefore we define three scores for a cluster k :

- $S_r(k)$: the regularity score. It is defined as the filling rate of the cluster grid. To compute this grid, we extract the maxima of the horizontal and vertical position distributions (see the red dots in Fig.3.4). These maxima are noted respectively x_1, \dots, x_p and y_1, \dots, y_q . Hence, the cluster grid is of size pq and the filling rate is: $S_r(k) = \frac{N_k}{pq}$.
- $S_g(k)$: the geometry score. It is computed from the horizontal and vertical jump distributions. The most probable jumps of these distributions in cluster k are noted j_X^k and j_Y^k , and the geometry score is defined as: $S_g(k) = \min(\frac{j_X^k}{j_Y^k}, \frac{j_Y^k}{j_X^k})$
- $S_m(k)$: the mass score. It is merely the number of points in the cluster N_k .

The total score of a cluster k is heuristically computed as the product of the three scores, from which the probability of a cluster k is derived:

$$p(k) = \frac{S_g(k)S_r(k)S_m(k)}{\sum_{k'} S_g(k')S_r(k')S_m(k')} \quad (3.9)$$

3.3.4 Global Position and Jump Distributions

Now that we have estimated empirically the probability of a given cluster k , we can compute the complete probability of X and J , by partitioning over the clusters. Hence, we have:

$$p(X = x) = p_X(x) = \sum_k p(X = x|k)p(k), \quad (3.10)$$

$$p(J = \gamma) = p_J(\gamma) = \sum_k p(J = \gamma|k)p(k) \quad (3.11)$$

These two probabilities contain the information gathered in the different clusters, and weights them according to the confidence we have in the different clusters. An example of such probability density functions are given in Fig.3.6.

3.4 Modeling Pattern Repetitions

In this section, a Markov chain models the repetitions of the pattern over the image horizontally and then vertically. The positions of the pattern as well as the relationship between two consecutive patterns are guided by the information extracted from the images. Since the problem is identical along the two principal directions, we will focus on the horizontal repetitions. The vertical repetitions are treated similarly. The use of a Markov chain formulation is indeed motivated by the

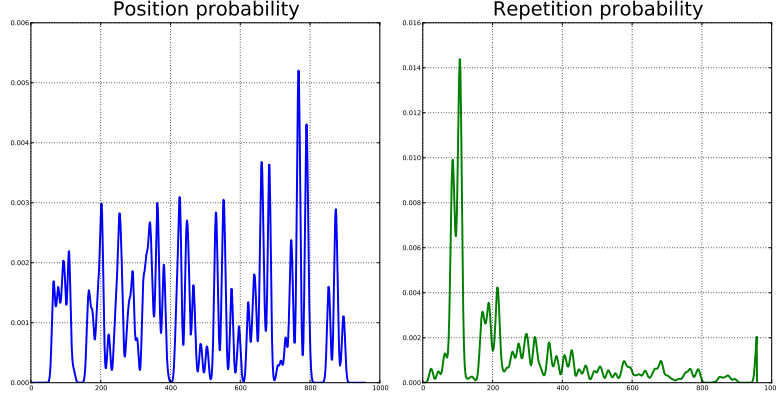


Figure 3.6: The final probability of jumps and positions of the pattern, as the summary of the information collected in the clusters.

grammar formulation. A simple facade grammar splits the facade vertically and horizontally, regularly or not. These two splits are here represented by two Markov chains that are optimized based on image evidence, gathered during clustering.

3.4.1 Markov Chain Formulation

Let W be the width in pixels of the image. Consider that at most n windows are expected horizontally. Let x_i be the random variable that represents the position of the i^{th} pattern. This position ranges potentially from 0 to W . We denote w_i the random variable representing the width of the i^{th} pattern. It can potentially be 0, meaning that the i^{th} pattern does not exist. Eventually we denote $z_i = (x_i, w_i)$ the random variable that completely describe the configuration of the i^{th} pattern.

We model the joint distribution of the n random variables using a Markov chain (see Fig.3.7). For numerical convenience, we express the join probability in terms of energy, using the Boltzmann transformation, or:

$$E(z_1, \dots, z_n) = \sum_{i=1}^n \phi_i(z_i) + \sum_{i=1}^{n-1} \psi_{i,i+1}(z_i, z_{i+1}) \quad (3.12)$$

where $\phi_i(z_i)$ is the unary potential with respect to the variable z_i and $\psi_{i,i+1}(z_i, z_{i+1})$ the pair-wise potential with respect to the variable z_i and z_{i+1} . The unary potential characterizes the appropriateness of having a pattern starting at position x_i and of width w_i . The pair-wise potential focuses on the joint configuration of two consecutive patterns. It penalizes patterns of with a distance inconsistent with the jumps found in the clusters. The choice of a Markov chain allows us to focus on consecutive patterns only. Thus, this boils down to study the gap between the patterns (see section 3.3.2). The definition of these potentials is detailed in the following subsections.

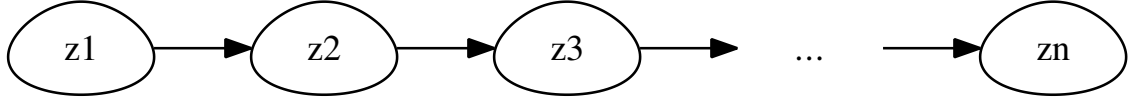


Figure 3.7: The Markov Chain formulation of the pattern repetition.

3.4.2 Unary Potentials

Let's define first the unary potentials. The value of $\phi_i(z_i) = \phi_i(x_i, w_i)$ is expected to be small when there is a high probability to find a pattern (a window) starting at x_i and spanning a width w_i . Therefore, we use the position distribution $p_X(x)$ defined in section 3.3.1. The probability of configuration (x_i, w_i) is big when there the segment $[x_i, x_i + w_i]$ corresponds to high values of p_X . Thus, we define it in the following way:

$$\phi_i(x_i, w_i) = \begin{cases} -\lambda \int_{x_i}^{x_i+w_i} p_X(t) dt & \text{if } 0 < x_i + w_i < w \\ \infty & \text{otherwise} \end{cases} \quad (3.13)$$

where λ is a factor used to balance the unary term with the pair-wise term. Note that the unary potential is more negative when the i^{th} pattern corresponds to a region of p_X with high values. Now let's define the pair-wise potentials.

3.4.3 Pair-wise Potentials

The pair-wise potentials $\psi_{i,i+1}(x_i, w_i, x_{i+1}, w_{i+1})$ should enforce two kinds of constraints. First, the two consecutive nodes z_i and z_{i+1} should be ordered, and non-overlapping; z_{i+1} should start after z_i ends. Thus, we enable the nodes to have dimension 0, but these "fake" patterns should lie after the existing ones. In other words, the fake nodes should be at position w only. Furthermore, the distance between two consecutive nodes should respect the jump distribution extracted in section 3.3.2. Besides, the energy of jumping from a node to the next one should be an increasing of the size of the jump. These pair-wise potentials aim at following the extracted jump distribution and to force at least some windows to exist (non-zero dimension). Another way to see the problem is the following. Imagine that we are trying to cross the image from left to right by jumping on the peaks of the position distribution, with jumps corresponding to the jump distribution. More formally, the pair-wise potential is defined by:

$$\psi_{i,i+1}(x, w, x', w') = \begin{cases} \infty & \text{if } x' < x + w \\ \infty & \text{if } w' > 0 \text{ and } w = 0 \\ \lambda \int_{x+w}^{x'} p_X(t) dt - \log(p_J(x' - x)) + \frac{(x' - x)^2}{\sigma^2} & \text{otherwise} \end{cases} \quad (3.14)$$

Moreover, it is possible to add a term to constrain the windows to have the same width, but it is usually not necessary, and even not interesting while dealing with the vertical split of the facade.

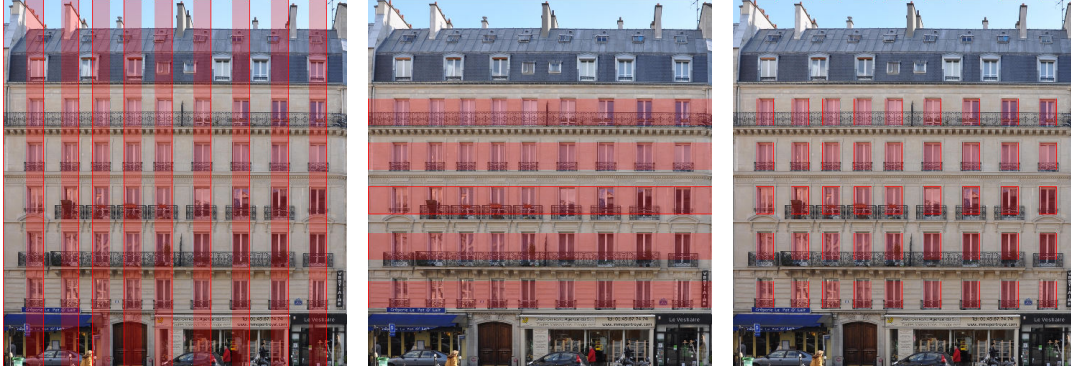


Figure 3.8: Representation of the two optimized parameters along X and Y as well as the final binary segmentation.

Eventually, σ is a characteristic jump, such as the most likely jump defined by p_J (see equation 3.11).

The suggested Markov chain energy addresses all the natural necessary conditions being imposed from the nature of a facade image. It consists of two sums, one over the whole length of the image of data terms and one over jumps. No matter the number of nodes in the chain, the energy always corresponds to these two sums over the size of the image. Therefore comparing such two energies is always coherent since they are defined on the same number of terms. Furthermore, the jump cost defined in the pair-wise potential is a convex term. Thus two small jumps have a smaller energy than a big one, provided that these jumps are themselves probable.

3.4.4 Optimization

In practice, the Markov chain is optimized with *Dynamic Programming* or equivalently *forward-backward propagation*. Both methods are equivalent and guarantee the optimality of the solution. Therefore, the seemingly continuous variables z_i need to be discretized. The possible values (x_i, w_i) for each variable z_i are called the labels. In order to get rid of clearly unpromising values we discard the labels with very unlikely positions x_i according to p_X , and keep a range of widths w_i .

3.5 Experimental Validation

Minimizing the energy defined in equation 3.12 for the horizontal and vertical cases provides two sets of optimal positions and sizes of the pattern: $\{(x_i^*, w_i^*)\}_{1 \leq i \leq n}$ and $\{(y_i^*, h_i^*)\}_{1 \leq i \leq n}$. They are then turned into a binary segmentation of the facade, in which the patterns cover the regions $\{[x_i^*, x_i^* + w_i^*] \times [y_j^*, y_j^* + h_j^*]\}_{1 \leq i \leq n, 1 \leq j \leq n}$. Let's now discuss and quantify the quality of these

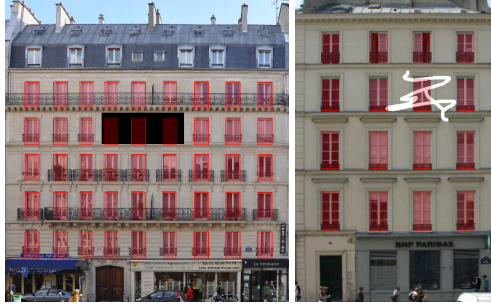


Figure 3.9: Examples of robustness to occlusions.

segmentations on different facade images.

3.5.1 Qualitative Results

First, we check the qualitative validity of this unsupervised method on several architectural styles. Fig.3.10 and Fig.3.11 show some examples of binary segmentations obtained by the proposed bottom-up parsing method. Note that the grids are not regular and that the facades are sometimes visually very challenging. Furthermore, the propose method is quite robust to occlusions as we can observe in Fig.3.9, where some parts of the image have been artificially occluded.

3.5.2 Quantitative Validation

To quantitatively evaluate the performances of the proposed bottom-up parsing method, we build a data set of 10 manually segmented buildings from New York City, USA. It gathers buildings of different sizes, with different scales, and varying number of floors and windows, and various styles. The proposed method is ran with exactly the same set of parameters on each of the images.

Dice Coefficient

We propose to quantify the quality of the obtained binary segmentations using two kinds of measures. The first one is the Dice coefficient of the segmentation with respect to the manual segmentations, considered as gold standard. It is computed for the corresponding 1D horizontal and vertical segmentations as well as the complete 2D binary segmentation. The Dice between two sets A and B is defined as:

$$D(A, B) = \frac{2|A \cap B|}{|A + B|} \quad (3.15)$$

In our case, A is the set of detected windows and B the set of windows defined by the ground truth. In this dataset, we obtain an average dice of 0.69.



Figure 3.10: Examples of building from different European cities. Styles, appearances, scales, rhythms of repetitions are different.

Topological Precision

The second evaluation criterion assesses the correctness of the retrieved number of patterns. We use a very simple measure of topological precision of the given segmentation. A pattern is supposed to be detected if its label in the Markov chain has a non-zero size: $w \neq 0$. Therefore, we can count the number of detected patterns n_d and compare it to the ground truth n_{gt} and compute the topological precision as:

$$\tau = \frac{|n_d - n_{gt}|}{n_{gt}} \quad (3.16)$$

	x	y	2D	std
Dice	0.801	0.862	0.691	0.105
Topology	0.904	1.00	0.904	0.113

Table 3.1: Dice Coefficients and topology precision averaged on the NYC dataset.



Figure 3.11: Examples of building from different American cities.

3.5.3 Failure Cases

Although this unsupervised method provides promising results in general, we notice that it also fails on some cases. These failures mainly happen when the repetitive pattern is quite complex and is itself made of several parts, like a window and a balcony. In those cases, it is common that the balconies are denser in features points than the windows. As a consequence, the detected pattern is eventually the balcony rather than the full window. On a specific image, finding a way to tackle this problem is feasible. Note that it is easy to add a heuristic penalty to either look for bigger patterns or smaller one. However, a good heuristic for a specific building is probably not a good heuristic for another one; getting rid of these failure cases is not straightforward in general. Thus, it seems that higher order structural constraints are needed in order to overcome the limitations of this bottom-up method.

3.6 Conclusion

In this chapter, we have reviewed the literature on bottom-up facade parsing and proposed a novel approach that gets rid of the usual assumption of regular repetitions. The proposed method detects interest points on the facade, clusters their SIFT descriptors and forms a set of 2D grids that contain all geometrical specification of the input building. Based on a shape grammar assumption, we propose a graphical model that decodes the evidence gathered in the clusters to recover the exact facade layout.

This method provides satisfying results on a large number of images, but still presents two main drawbacks. First, it provides a very limited binary segmentation of the image, which is often not enough while aiming at performing realistic image-based modeling of facades. The second drawback is shared with most of the methods in the literature. The very appealing property that the segmentation is unsupervised and is built only upon image cues hides some lack of robustness and some necessary heuristics (size of the elements and their expected repetitions). By doing these extra assumptions, it is easier to obtain better results on a large number of facades, but it also limit

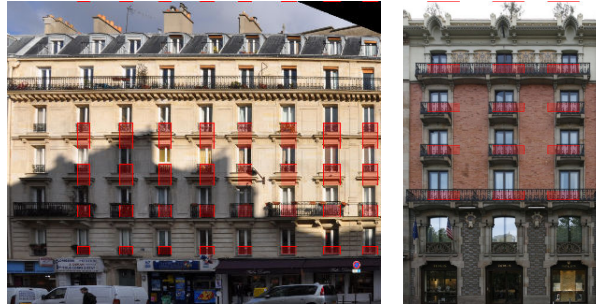


Figure 3.12: Failure Cases: the balconies are stronger than the windows, and the detected pattern sticks to them.

the scope of the method. Besides, such assumptions can be seen somehow as a first step towards proper facade modeling.

Eventually, a last drawback of the method is somehow the lack of control on the solution. As stressed in the failure cases, the obtained segmentation is sometimes very far from the expected one, and does not look like a facade. This is partly due to the Markov Chain model that does not restrict much the repetitions of the pattern.

Therefore, it seems to be more promising to force the segmentation to be driven by a shape grammar as presented in chapter 2. This approach is both more appealing and more challenging. On the one hand, it would enforce more control on the segmentation and would guarantee better quality of parsing with potentially more than two classes. Such an approach would perform a real multi-class semantic segmentation of building facade. On the other hand, top-down parsing is theoretically more challenging and practically heavier to implement. In the two next chapters, we will provide a novel framework for top-down parsing based on Reinforcement Learning and explain its theory and its application to complex facade segmentation and image-based procedural modeling.

Chapter 4

Top-Down Parsing: A Novel Algorithm

This chapter contains the main contribution of the thesis. We present here a stochastic algorithm to parse a specific type of shape grammars called BSG (see chapter 2). Parsing a shape grammar consists in finding the sequence of grammar rules that best explains the image. In this chapter, this algorithm is defined and applied on artificial data so as to get a better understanding of its mechanisms. Applications to real data will be presented in chapter 5.

After reviewing the state-of-the-art of grammar parsing in section 4.1, we introduce the theoretical framework of Markov Decision Processes in section 4.2. The parsing algorithm is explained and studied in section 4.3, while section 4.4 shows how Reinforcement Learning allows us to go further, supporting function approximations, model selection, user interactions and data-driven exploration.

4.1 State of the Art

Formal grammars have been studied for the last 55 years starting with the work of Noam Chomsky [Chomsky 1956]. Among the three models described by Chomsky, one received a particular attention from the computer science community: Context-free grammars (CFG). Indeed they are a good trade-off between complexity and expressive power. The definition of a context-free grammar was given in section 2.1.

A context-free grammar can produce many sentences (made only of terminal symbols). Knowing a grammar, one of the first questions that naturally comes up is the following: can we decide if a given sentence belongs to the language of our grammar, and if so, which sequence of rules leads to it? This problem is called *parsing*, and aims at discovering the implicit grammatical structure of a sentence. The parsing problem can be extended to probabilistic context-free grammars (PCFG), where the replacement rules are associated with probabilities. When several parse trees explain the input sentence, the best parse is the one associated to the highest probability. PCFGs are widely used in Natural Language Processing (NLP), where the optimal parse informs about the nature of the different words and group of words in the sentence (verbs, nouns, subjects, complements, etc.).

In our case though, the sentences are 2D images, but the principle of parsing remains. The question is now: which sequence of rules best explains our image? In order to tackle this very complex problem, let's first review the literature of string grammar parsing and the attempts of shape grammar parsing in Computer Vision, before entering the details of the proposed algorithm.

4.1.1 Context-Free Grammar Parsing

There are basically two classes of parsing algorithms: bottom-up or top-down. Top-down parsing methods start from the axiom, and go down to the terminal nodes by applying grammar rules and expanding the parsing trees until it reaches the leaves (input). On the contrary, bottom-up algorithms start from the leaves of the parse tree and recursively build up the parent nodes by gathering current leaves according to the right-hand sides of the rules.

In both cases, it is interesting to notice that these methods share two common properties: dynamic programming (DP) for optimization and Chomsky Normal Form (CNF) as a representation of the grammar. This is the case of two of the most famous algorithms: Earley's parsing [Earley 1970] (top-down) and Cocke-Younger-Kasami [Younger 1967] or CYK (bottom-up). Both can be adapted to probabilistic context-free grammars. These two properties will somehow be extended to the split grammar parsing in section 4.3. These DP based methods are very generic and of complexity $O(n^3)$ with n being the size of the parsed string. There exist faster parsing methods (in $O(n)$) but that are valid only on specific subclasses of CFGs (LR and LL types of parsing).

4.1.2 Image Grammar Parsing

In Computer Vision, image parsing has been a very active field from the beginning. It aims at understanding the structure of an image, its semantic regions as well as the relationship between them. In the 1970's, the parsing problem was already set by Ohta and Kanade [Ohta 1978]. This visionary paper sets up the fundamental notions of hierarchical representation, semantic segmentation and top-down/bottom-up approaches. Image parsing somehow adds a semantic layer to image segmentation. Bridging the so-called *semantic gap* between raw pixels and a symbolic representation of images still constitutes the Holy Grail of Computer Vision. The problem is intrinsically much more complex than the NLP one for many reasons. An English sentence is by nature discrete, sequential and clearly identifies its words. On the contrary, images are made of unidentified continuous 2D objects. Therefore we are facing two problems in image parsing: the curse of dimensionality on the one hand, and object detection/recognition on the other hand. The same object never has the same appearance in two images (viewpoints, illumination, occlusions, etc.), and defining object categories is a very complex task still under investigation in Computer Vision due to the infinite intra-class variability of object appearances.

Even if the Computer Vision community is recently reconsidering the power of Dynamic Programming [Felzenszwalb 2010a, Felzenszwalb 2010b], the dimensionality of the parsing problem is too high to be solved with simple Dynamic Programming approaches. DP suffers the famous *curse of dimensionality*. For that matter, three main classes of optimization paradigms have been

proposed to solve image parsing: Markov/ Conditional Random Fields (MRF and CRF), variants of Markov Chain Monte-Carlo, and pruned variants of dynamic programming itself and A* algorithm.

Conditional Random Fields and Markov Random Fields can be efficiently optimized (under some constraints) and therefore were quite successfully used in image parsing [Berg 2007, Shotton 2007, Tighe 2010]. They succeed in combining classification at the pixel level (or super pixel in [Tighe 2010]) with spatial regularizations among them. However, these random fields perform quite badly at modeling: they only provide a flat representation of the image. Hierarchical models have always been more attractive while making the inference much more sophisticated. Zhu and al. [Zhu 2000, Tu 2002] propose a data-driven Markov Chain Monte Carlo (DDMCMC) for image parsing, where segmentation proposals are image-driven. Using bottom-up cues drastically speeds-up the convergence of the algorithms. Zhu and al. [Zhu 2006] generalize the classic parse trees and propose a parsing graph. It is constructed and then dynamically modified by a reversible jumps Markov Chain Monte-Carlo, integrating top-down and bottom-up inferences to perform proper hierarchical image segmentation. AND/OR graph is another hierarchical representation that provides good modeling and inference capabilities. Chen and al. [Chen 2007b] optimize it in a top-down/bottom-up process for object detection and segmentation, whereas Felzenszwalb and Mcallester [Felzenszwalb 2007] use A* algorithm. A* is a generalization of the greedy Dijkstra's algorithm for shortest path finding. The idea is to use a heuristic as a lower bound of the real shortest path. A* and DP are closely related. This algorithm was already introduced in PCFG parsing by Klein and al. [Klein 1994]. Hierarchical object detection is another example where A* performs very well as shown by Kokkinos and Yuille [Kokkinos 2009]. Dynamic Programming itself has also been used in image parsing, but using grammars with limited vocabulary or depth of derivation. In [Zhu 2008], a set of 30 generic segmentation patterns are used in a 3 layers hierarchical modeling for image parsing, using DP for inference. In [Chen 2007a], the authors first tackles the task of learning a probabilistic grammar, and then use it for parsing based on DP.

The generic image parsing task is undoubtedly extremely difficult and therefore people have tried to parse more constrained scenes with more constrained image grammars. Buildings offer a good trade-off for this problem. They show huge intra-class variability but at the same time have specific geometric constraints. As a consequence a grammar is well-suited to describe them and data are easy to get. This last point is not as trivial as it may seem. For instance, image grammars have been proposed to model biological structures in microscopic data [Schlecht 2007] which are more difficult to acquire than buildings. For those reasons, building facade parsing constitute a very good parsing problem, and solving it may bring answers to the generic image parsing quest [Zhu 2007].

4.1.3 Facade Parsing

Facade parsing seems to be a very specific task, but turns out to have received a huge attention over the past few years. This rocketing interest is due to the very attractive applications in virtual city reconstructions for video games, cinema industry or web applications. There are basically two classes of methods to tackle this problem: unsupervised bottom-up approaches discussed in

chapter 3 and top-down grammar-based parsing that we are going to review now. Pure bottom-up approaches try to discover repetitions along vertical and horizontal directions so as to get the coarse grid layout of the facade and sometimes refine some of its elements. On the contrary, top-down methods try to impose a complex shape prior that matches bottom-up cues. These methods usually differ according to the complexity of their shape prior and the way to encode these high order constraints in the optimization framework.

Alignments are the first shape priors people have attempted to enforce on building images. Dick and al. [Dick 2000, Dick 2002] describe buildings by a kind of Lego with parameterizable parts for which the number of parts is itself unknown. Using automatically calibrated multiple views as inputs, and learned texture parameters for each type of elements (windows, etc.), the optimal structure of the building is computed using a reversible jumps Markov Chain Monte-Carlo (rjMCMC) that enables them to encode the shape prior. This formulation is somewhat close to a grammar formulation but remains flat. MCMC is very popular in this field. Reznik and al. [Reznik 2007] use it coupled with Implicit Shape Models, and Korah and al. [Korah 2008] make use of MCMC to fit a grid of rectangles. These approaches are still providing a flat modeling of the building facade. A last interesting flat approach has been proposed by Cech and al. [Cech 2008, Cech 2009]. They manage to encode 2D binary grammars in a MRF formulation by building a label set that not only contains the semantic interpretation (wall or window) of the pixels but also the relative positions of the elements (left/right/up/down). They define strong structural potentials based on these augmented labels that in the end enforce alignments. However, this elegant formulation fails at being extended to more complex grammars.

Hierarchical models based on more complex 2D grammars than simple grids have also been used and optimized using rjMCMC. Alegre and al. [Alegre 2004] propose the first solution to probabilistic context-free grammar parsing for facades. Like many other works, they process rectified images. Besides, they assume the elements to be rectangles with uniform color. This strong assumption somewhat limits the scope of their work, but the method is easily extensible to more complex images measurements. rjMCMC enables them to optimize the topology of the parse tree as well as its parameters. [Ripperda 2007, Ripperda 2009] also optimize a PCFG with rjMCMC, and score the parse trees based on a more complex pixel level function and a minimum description length (MDL) criterion, encouraging simple models.

Eventually, a last grammar-based approach with Monte-Carlo like method has been proposed by our group [Teboul 2010]. This work uses complex split grammars made of n terminal shapes, with deep levels of derivation. The appearance model of each terminal shape has been learned in an off-line training step using Randomized Forest [Breiman 2001]. The learning step provides for each pixel x of the image \mathcal{I} and each semantic class c (wall, window, door, etc) the posterior probability $p(c|x, \mathcal{I})$. A specific derivation of the input corresponds to a specific shape C of the language $L(G)$. In the image, it corresponds to a specific image segmentation, providing a label $C(x)$ to each pixel x . We defined an energy to quantify the quality of a grammar-based segmentation:

$$E(C) = - \sum_{x \in \mathcal{I}} \log p(C(x)|x, \mathcal{I}) \quad s.t \ C \in L(G) \quad (4.1)$$

The goal of the proposed method is to optimize $E(C)$ with respect to C . Since the grammar constraint is difficult to catch, a natural way to enforce it is to follow the derivation so as to be sure to keep $C \in L(G)$. Starting from a regular initial seed, the parameters of each rules are randomly resampled one at a time around the current seed by a perturbation model. The best instances are sequentially kept until convergence. The global derivation scheme is fixed, but the modification of the rules' parameters may impact on the real topology of the building on the image. This method gives good results, but deals with a fixed derivation scheme, and turns out to be quite slow. The main reason is that the building model is scored as a whole, and locally bad decisions are not directly detected as so. In this chapter, we will not present any further this previous work that is considered as part of the state-of-the-art, and we will rather focus on a more important work that introduces a better parsing framework for split grammars. These grammars indeed encompass all the grammars that were already used in the related work presented above.

One of the main contributions of this thesis comes from a change of viewpoint in the facade parsing problem. Parsing a probabilistic image grammar means to find the sequence of rules that best explains the image. A sequence of rules can be seen without loss of generality as a sequence of decisions: which rule to apply next. Therefore, parsing an image is equivalent to solving a decision process. We will see that this decision process has special properties that makes it a (semi) Markov Decision Process (MDP), opening the door to a whole class of optimization algorithms called *Reinforcement Learning* and that has as two extreme cases Dynamic Programming and Monte-Carlo.

4.2 A Brief Overview Markov Decision Processes

Computer Vision is the playground of Machine Learning. Indeed, many machine learning problems have found good applications in Computer Vision since the image data usually provide the expected complexities and scales for these algorithms. Although *Supervised Learning* and *Clustering* have been very popular and successful in this field, *Reinforcement Learning* (RL) has surprisingly received almost no attention among the community.

Supervised learning is learning from examples. It often takes the form of classification. One can train a classifier to learn how to recognize hand-written digits, faces, etc. by feeding it with annotated examples. The most famous methods are boosting, support vector machines and randomized forests. Unsupervised learning aims at learning the geometry of a dataset by grouping similar data into clusters. Reinforcement Learning follows a more natural learning principle: learn from interactions with the environment. It seems to be closer to the way life learns. A child learns how to walk by trying to walk. As a consequence, Reinforcement Learning is goal-oriented more than the other machine learning paradigms.

Reinforcement Learning encompasses different kinds of methods such as optimal control, automation or artificial intelligence, which were gathered recently since they are different ways to solve the same problem: optimizing a decision process. This class of algorithms was used in many different applications such as control of chemical processes, games [Tesauro 1995] or robotics.

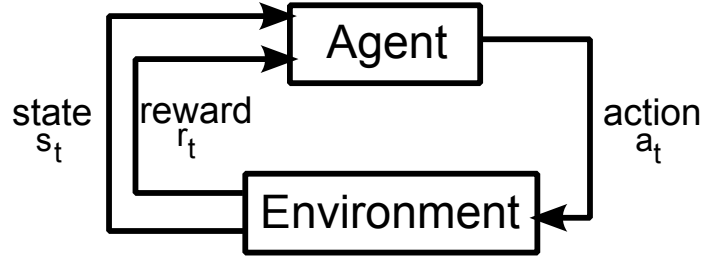


Figure 4.1: Agent-environment interaction scheme.

Sometimes, RL is also referred as Neuro-Dynamic Programming [Bertsekas 2007] for its relationship with dynamic programming.

Let's first study the Markov Decision Processes which define the Reinforcement Learning problem before analyzing how to formulate our facade parsing problem within this framework. In the remainder of the chapter, we will use the same notations as in the reference book in Reinforcement Learning by Sutton and Barto [Sutton 1998].

4.2.1 Markov Decision Processes

Richard Bellman [Bellman 1957b, Bellman 1957a] and Ronald Howard [Howard 1960] first studied Markov Decision Processes (MDPs) and described the interaction between an *agent* and its *environment*.

Definitions

We describe a MDP by a set of states \mathcal{S} , a set of actions \mathcal{A} , transitions probabilities between states P and expected rewards consecutive to the actions R . The interaction loop is illustrated in Fig.4.1.

This figure can be interpreted that way. At time t , the environment sends a state signal $s_t \in \mathcal{S}$ to the agent. Based on s_t , the agent takes the action $a_t \in \mathcal{A}(s_t)$. This action modifies the environment, which sends back to the agent a new state signal (or state in brief) s_{t+1} as well as an immediate reward r_{t+1} . The transition from a state s to a state s' consequent to an agent's action a is subject to a probability $P_{ss'}^a$. The reward received by the agent after being in state s , playing action a and arriving in state s' has an expectation $R_{ss'}^a$. More formally, P and R are defined as:

$$P_{ss'}^a = p(s_{t+1} = s' | s_t = s, a_t = a) \quad (4.2)$$

$$R_{ss'}^a = \mathbb{E} [r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'] \quad (4.3)$$

On top of that, a Markov Decision Processes is subject to the *Markov Property*. The transition probabilities from a state to another and the immediate rewards only depend on the current state

and the chosen action and not on the past states. As a consequence, we will see later that the dynamics of an MDP is governed by a one-step recursion, linking a state to the next possible ones. The Markov property is formally formulated as:

$$p(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, \dots, s_0, a_0) = p(s_{t+1} = s', r_{t+1} = r | s_t, a_t) \quad (4.4)$$

Therefore, an MDP is completely defined by a 4-tuple $(\mathcal{S}, \mathcal{A}, P, R)$. We are now going to define the agent's goal and see later how the Markov property allows us to formulate a one-step recursive equation known as the Bellman equation.

Reinforcement Learning Goal

The goal of the agent is to maximize his long-term reward, also called the *return*, that is to say the amount of rewards he will accumulate until the end of the process. Please note that a decision process can be infinite, and therefore there could be no end. The number of steps or decision tasks is called the horizon T , and can be finite or infinite. In case of finite horizon MDPs, we call *episode* a sequence of states from an initial one to the end.

In order to properly define the return, we introduce a discount γ , and define in a very generic way the return at time t as the power series (Maclaurin series) of the discount variable associated to the rewards sequence:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (4.5)$$

In our case, we will restrict ourselves to undiscounted ($\gamma = 1$) finite MDPs ($T < \infty$). The goal of the agent is therefore to maximize R_t for all t . Before going any further, we first have to introduce the concept of policy.

Agent's Policy

A *policy* is a key in solving MDPs since a policy π basically describes the behavior of the agent. Therefore, solving a MDP is equivalent to finding an optimal policy. More formally, a policy π is a mapping from each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$ to a probability, the one of choosing action a while being in state s .

$$\pi(s, a) = p(a|s) \quad (4.6)$$

As we discussed earlier, the goal of the agent is to take the decisions that will guarantee him to maximize its expectation of gain in the long-run. Rephrased in the Reinforcement Learning framework, an agent is looking for a policy that maximizes the expectation of the returns. This leads naturally to the question of evaluating the quality of a policy with respect to the agent's goal.

Value Functions and Bellman's Equation

The *state-value function* $V^\pi(s)$ is exactly answering the question of measuring the quality of a policy. It is defined as:

$$V^\pi(s) = \mathbb{E}_\pi [R_t \mid s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right] \quad (4.7)$$

Indeed $V^\pi(s)$ represents what the agent can expect to gain in the long-run when starting in state s and following the policy π thereafter.

Similarly, we can define an *action-value function* $Q^\pi(s, a)$ that represents what the agent can expect in the long-run, while starting in state s , taking action a and following the policy π thereafter. Following the policy π means that the agent chooses an action a when in state s with a probability $\pi(s, a)$. The action-value function is naturally defined as in equation 4.8:

$$Q^\pi(s, a) = \mathbb{E}_\pi [R_t \mid s_t = s, a_t = a] \quad (4.8)$$

Sometimes, we will merely refer to these functions as the V-function and the Q-function. Obviously there are some links between these two functions. $V(s)$ can be seen as a combination of Q weighted by the probability $\pi(s, a)$ to choose action a while in state s . And since taking action a when in s leads to s' with probability $P_{ss'}^a$ and gives an expected reward $R_{ss'}^a$, we easily get these two relationships:

$$V^\pi(s) = \sum_a \pi(s, a) Q^\pi(s, a) \quad (4.9)$$

$$Q^\pi(s, a) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')] \quad (4.10)$$

Combining them leads naturally to the Bellman equation for the state value function:

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')] \quad (4.11)$$

This equation states the relationship between the value of the V-function at state s and the V-function at all possible next states s' . It is a linear equation in $V^\pi(s)$. Gathering all the equations for all the states, we get a system of linear equations that, in theory, can be inverted to get V^π . This function quantifies the quality of a known policy π . In practice, however, we will see in section 4.2.2 how to solve this problem of *policy evaluation*.

Optimal Policies

Remember that we want to find the optimal policy for the MDP. So far, we have discussed about how to quantify a policy. Therefore we can now compare a policy to another one, in order to identify the best policy.

A policy π is better than a policy π' for a given MDP, if:

$$\forall s, \quad V^\pi(s) \geq V^{\pi'}(s) \quad (4.12)$$

Since the decision process satisfies the Markov property, there is always at least one policy that is better than the other ones. We call it the *optimal policy* and naturally denote it π^* , such as:

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi} V^\pi(s), \quad \forall s \quad (4.13)$$

$$Q^*(s, a) = Q^{\pi^*}(s, a) = \max_{\pi} Q^\pi(s, a) \quad (4.14)$$

Here again we have the relationships between V^* and Q^* :

$$V^*(s) = \max_a Q^*(s, a) \quad (4.15)$$

$$Q^*(s, a) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^*(s')] \quad (4.16)$$

Combining equations 4.15 and 4.16 leads to the two Bellman optimality equations, which are not linear anymore:

$$V^*(s) = \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^*(s')] \quad (4.17)$$

$$Q^*(s, a) = \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma \max_{a'} Q^*(s', a') \right] \quad (4.18)$$

Note that the optimal policy π^* does not appear in these equations; it is implicitly deterministic. Therefore, computing the optimal policy π^* is equivalent to computing Q^* . An interesting property of the optimal policy π^* , is that being greedy with respect to π^* guarantees an optimal return. Therefore the optimal policy solves the Markov decision processes. At each state s , the agent must choose the action that maximizes $\pi^*(s, \cdot)$, or equivalently the action that maximizes $Q^*(s, \cdot)$ so as to maximize its expectation of gain in the long-run.

Now that we have defined the key concepts of states, actions, rewards, returns, policies, value functions and optimality, the natural question that remains unanswered is how to solve an MDP in practice: how to compute the optimal policy?

There exist basically three classes of methods to solve a Markov Decision process that are all iterative. Dynamic Programming (DP) uses directly the Bellman optimality equation as an update equation. Monte-Carlo (MC) methods sequentially sample actions along the process to explore one full trajectory of the state space. Somewhere in between, Reinforcement Learning (RL) methods are combining the MC sampling and the DP update in an elegant framework. We will describe them all in the next sections.

It is important to notice that these three classes of methods share a common generic scheme called *Generalized Policy Iteration*. The idea is to start from a policy π_0 and keep iterating between two steps:

Algorithm 4.1 Value Iteration for a Deterministic MDP

```

Initialize randomly  $V(s), \forall s \in \mathcal{S}$ 
repeat
   $\Delta \leftarrow 0$ 
  for  $s \in \mathcal{S}$  do
     $V_0 \leftarrow V(s)$ 
     $V(s) \leftarrow \max_a [R_{ss'}^a + \gamma V_k(s')]$ 
     $\Delta \leftarrow \max(\Delta, |V(s) - V_0|)$ 
  end for
until  $\Delta < \text{threshold}$ 

```

1. Policy Evaluation by estimating one of the values functions (V or Q)
2. Policy Improvement, based on the current estimate of V or Q .

That being said, let's go a bit more into the detail of DP, MC and RL and discuss their advantages and drawbacks.

4.2.2 Dynamic Programming

Dynamic Programming is a family of algorithms solving MDPs that is guaranteed to converge towards the optimal solution. The general idea of these methods is based on the generalized policy iteration scheme. Here we briefly describe the most famous algorithm called *value iteration*, and discuss about its limitations and its relation with what computer scientists usually refer as DP.

Value Iteration

This algorithm turns the Bellman optimality equation 4.18 into an update rule for the estimate of the V-function to iteratively estimate $V^*(s)$ directly for all states s . Thus, the estimates of $V^*(s)$ are computed based on estimates of $V^*(s')$.

$$V_{k+1}(s) = \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')] \quad (4.19)$$

The idea of the algorithm is to traverse the state space \mathcal{S} and keep updating the current estimate $V_k(s)$ until they have all converged. Then, the optimal policy is the greedy policy with respect to Q^* that we can easily compute from $V^*(s)$ with equation 4.15 and equation 4.16. We call a greedy policy with respect to Q a deterministic policy such that $\pi(s, a^*) = \delta_{a^*}(\arg \max_a Q(s, a))$, where δ represents the Kronecker function. The complete value iteration algorithm is given in algorithm 4.1 in the case of deterministic MDP. Of course the algorithm can be adapted to stochastic case by simply replacing the update as in equation 4.19.

Limitations

Value iteration (and by extension DP methods in general) has two main limitations.

Complete knowledge of the environment P and R The use of equation 4.19 requires having a complete knowledge of the environment. This is not always the case. In this chapter, we will restrict ourselves to *deterministic* decision processes. In such a process, the transitions between states are not stochastic anymore, the knowledge of the state s and the action a completely determines the next state s' . However, we will see that some stochasticity may remain in the expected rewards.

Curse of dimensionality The agent has to perform complete *sweeps* through the state space. However, this state space is often very large, and can even be infinite while dealing with continuous states. As a consequence, DP methods are often extremely slow, require insane amounts of memory, or are restricted to small state spaces.

Furthermore, while traversing the whole state space, the agent spends most of his time in very uninteresting states. For that matter, extensions to classic DP have been proposed to modify the order in which the states are visited, this class of methods called *prioritized sweeping* has been proved to converge [Li 2008].

Relation with Dynamic Programming outside of the MDP framework

In Computer Science, Dynamic Programming is a well-known class of optimization methods. One may wonder what the link between DP, as presented in MDPs, and DP, as computer scientists usually expose it, is.

For that matter, let's take a very classic optimization problem: find the minimum of a multi-variate function cost f of the variables $(x_i)_{i \leq n}$ of the form:

$$f(x_1, \dots, x_n) = \sum_{i=1}^n \phi_i(x_i) + \sum_{i=1}^{n-1} \psi_{i,i+1}(x_i, x_{i+1}) \quad (4.20)$$

This is a specific case of a more general class of functions or energies very popular in Computer Vision. ϕ_i represents a data term, and $\psi_{i,i+1}$ a pairwise term between neighboring variables that is usually used as smoothness or regularization term. This function can be represented by a Markov chain.

The basic idea to solve such a problem is to say that the solution in Dynamic Programming is made of partial solutions (for instance, a shortest path is made of shortest paths). We basically store these partial solutions to the problem, or *cost-so-far*, into a table B , such as:

$$\begin{aligned}
B_1[x_1] &= \phi_1(x_1) \\
&\vdots \\
B_i[x_i] &= \phi_i(x_i) + \min_{x_{i-1}} (\psi_{i-1,i}(x_{i-1}, x_i) + B_{i-1}[x_{i-1}])
\end{aligned} \tag{4.21}$$

$B[i]$ represents the optimal cost-so-far for all the possible values of x_i , or cost from the first variable to the variable x_i , for all the possible x_i . As a consequence, the minimal cost for the entire sequence x_1, \dots, x_n is exactly the minimal entry of B_n . Back tracking the sequence, we obtain the values (x_1^*, \dots, x_n^*) that minimize f :

$$\begin{aligned}
x_n^* &= \arg \min_{x_n} B_n[x_n] \\
&\vdots \\
x_i^* &= \arg \min_{x_i} [B_i[x_i] + \psi_{i,i+1}(x_i, x_{i+1}^*)]
\end{aligned} \tag{4.22}$$

Since the problem is symmetrical, we could have solved the optimization task by inverting the variable order, and compute a cost-to-go $\tilde{B}[i]$ representing the cost to go from x_i to the end:

$$\begin{aligned}
\tilde{B}_n[x_n] &= \phi_n(x_n) \\
&\vdots \\
\tilde{B}_i[x_i] &= \phi_n(x_i) + \min_{x_{i+1}} (\psi_{i,i+1}(x_i, x_{i+1}) + \tilde{B}_{i+1}[x_{i+1}])
\end{aligned} \tag{4.23}$$

Now we can actually reformulate this Dynamic Programming algorithm using the MDP framework presented previously. We consider the undiscounted ($\gamma = 1$) deterministic finite horizon ($T = n$) Markov Decision Process such that:

States represent the variables: $s_t = x_t$

Actions represent the possible next states (deterministic): $a_t = x_{t+1}$

Rewards are directly linked to the costs: $R(s_t, a_t) = -[\phi_t(x_t) + \psi_{t,t+1}(x_t, x_{t+1})]$
By convention, we set that $\psi_{n,n+1}(x_n, x_{n+1}) = 0$.

V-function is what the agent can expect to get until the end of the process, therefore it does corresponds to the cost-to-go \tilde{B} .

Value Iteration Since the process is deterministic, the rewards are known and we are facing a finite task, the update equation is:

$$\begin{aligned}
V(s) &= \max_a [R_{ss'}^a + V(s')] \\
&= \tilde{B}[x_i] = -\phi_i(x_i) - \max_a [\phi(x_i, x_{i+1}) + \tilde{B}[x_{i+1}]]
\end{aligned} \tag{4.24}$$

As a consequence, this simple example shows how classic DP can be rephrased in the MDP framework. However, the MDP framework remains more generic than classic DP formulation, since it supports stochastic transitions and stochastic rewards.

Ultimately, we would like to point out here that solving this very famous problem with DP, is also equivalent to the Viterbi parse or the min-sum algorithm on a Markov chain. Now that we have understood how DP can find the optimal policy for a generic MDP, let's explore another family of solution known as the Monte-Carlo methods.

4.2.3 Monte-Carlo

Monte-Carlo (MC) methods follow another spirit to solve a Markov Decision Process. MC methods have been widely used in many fields outside of the MDP framework. Historically they were first used to estimate expectations (and therefore integrals), and then expanded to many applications. The key idea of MC methods is *sampling*. Dynamic programming keeps updating the estimates of the V-function based on all the next estimates at all the possible next steps. MC takes the opposite strategy. The idea is to sample only one action at each time step according to the current policy. At the end of the episode, a sample of the return for each of the visited state is available. Consequently these experienced returns are used to update the estimates of Q according to the current policy along the sampled trajectories. Following the generalized policy iteration scheme, we alternate between policy evaluation and policy improvement. An example of MC algorithm is given in algorithm 4.2. This algorithm computes the expectation of return for each state-action pair s, a . This mean is formulated in an iterative way by the following update equation:

$$Q_{k+1}(s, a) = \frac{n_k(s, a)Q_k(s, a) + R(s, a)}{n_k(s, a) + 1} \tag{4.25}$$

$$= Q_k(s, a) + \frac{1}{n_k(s, a) + 1} [R(s, a) - Q_k(s, a)] \tag{4.26}$$

where $n_k(s, a)$ represents the number of times the pair (s, a) has been visited in the past and $R(s, a)$ the experienced return, or cumulative reward from s, a until the end of the episode. This kind of update equation is standard in Reinforcement Learning. It consists in iteratively estimating the expectation of a random variable Q by moving the current estimate Q_k towards the new observation R .

Comparing algorithm 4.1 to algorithm 4.2, we first notice that the MC algorithm gets rid of the two limitations mentioned in the previous section: the curse of dimensionality and the necessity to have a complete knowledge of the environment dynamics (R and P). Indeed, estimating the

Algorithm 4.2 Monte-Carlo method for solving MDP

```

Initialize randomly  $\pi$ 
 $Q(s, a) = 0, \forall s, a$ 
Initialize the number of visits  $n(s, a) = 0, \forall s, a$ 
loop
  Sample an episode according to  $\pi$ 
  for  $(s, a) \in$  episode do
     $R \leftarrow$  the return from  $(s, a)$ 
     $n(s, a) \leftarrow n(s, a) + 1$ 
     $Q(s, a) \leftarrow Q(s, a) + \frac{1}{n(s, a)}[R - Q(s, a)]$ 
  end for
  for  $s \in$  episode do
    update  $\pi$  according to  $Q(s, \cdot)$ 
  end for
end loop

```

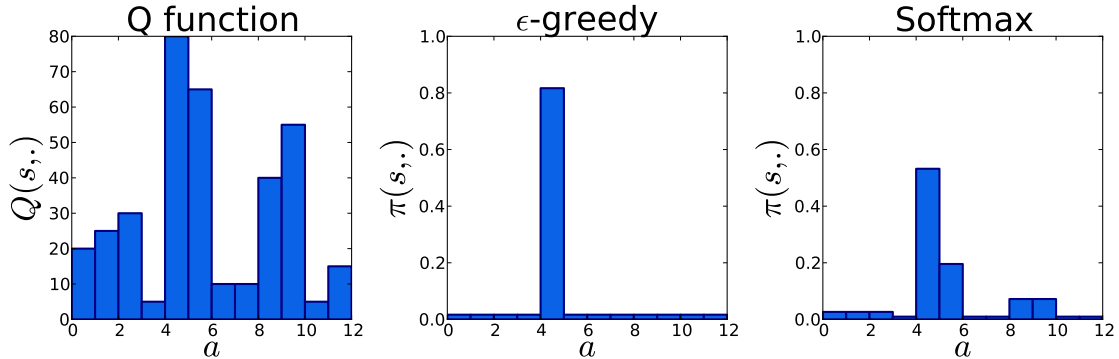
action-value function Q is very convenient while the environment is unknown. Besides, sampling is compatible with this assumption. We replace here *knowing* the environment, by *interacting* with it. This observation is crucial in Reinforcement Learning. A good way to optimize a decision process is based on *trial-and-errors*. Since we do not know the environment, we test it instead. Although sampling seems to control the curse of dimensionality, it also raises the question of the convergence. Another question raised by MC method is the one of policy improvement. Algorithm 4.2 states that after each episode, the current policy should be updated according to the current estimate of Q for all the visited states. Which probability distribution to choose for π at these states? Let's first study the possible ways to update a policy and then discuss about the convergence of MC. The classic policy families we are going to introduce now will also be used in pure Reinforcement Learning algorithms.

Policy Improvement

There are two classic ways of adapting a policy according to the Q-function. They are called *ϵ -greedy* and *softmax* policies. Recall first that a greedy policy with respect to the Q-function would be a deterministic policy. That is to say a policy where one action, called the greedy action would have probability 1 and all the other will have probability 0. We keep calling the *greedy action* the one associated to the highest probability in the policy. It should also correspond to the action with the highest expected gain at state s . In other words, if we have an estimate of the action-value function Q , then we define the greedy action a^* as:

$$a^* = \underset{a}{\operatorname{arg\,max}} Q(s, a) \quad (4.27)$$

Improving the policy by making it greedy with respect to the current estimate of Q might be

Figure 4.2: ϵ -greedy and softmax policies with respect to Q .

dangerous for the convergence of the algorithm, since the current estimate of Q might be wrong, especially at the beginning of the loop, where few episodes have been sampled. For that reason, we need to let some room for *exploration*. This possibility to choose seemingly less trustworthy action is a very common problem in evolutionary algorithm called *the exploration-exploitation trade-off*. It means that these algorithms have to find a balance between exploiting the knowledge they have built and exploring new regions of the state space. This balance should also evolve into time, so that the agent explores more at the beginning when he has a poor knowledge of its environment, and explore less and less as iterations go. The more the agent knows his environment, the more he trusts his knowledge. We now give two examples of possible policy families.

ϵ -greedy policy This policy is designed to be almost greedy. Let's suppose that the agent is in state s , and is facing a choice among $n = |\mathcal{A}(s)|$ possibilities. Then, its policy is ϵ -greedy if with probability $1 - \epsilon$ the agent chooses the greedy action a^* and with probability ϵ the agent chooses uniformly any action. More formally the policy is defined by:

$$\pi(s, a) = \left(1 - \epsilon + \frac{\epsilon}{n}\right) \delta_{a^*}(a) + \frac{\epsilon}{n} [1 - \delta_{a^*}(a)] \quad (4.28)$$

Usually the exploration rate ϵ decreases to zero as the agent learns, according to a predefined law. An example of $Q(s, \cdot)$ and the corresponding ϵ -greedy policy for $\epsilon = 0.2$ are given in Fig.4.2.

Softmax policy The previous policy clearly encourages the agent to follow the greedy action. However, there might be cases where the second best action is indeed almost as good as the greedy one, and therefore it might be a good idea to give it more credit than the worst estimated action. For that reason, the softmax policy uses a Boltzmann distribution with “temperature” t_0 . In state s , the agent picks action a with probability:

$$\pi(s, a) = \frac{e^{\frac{Q(s,a)}{t_0}}}{\sum_b e^{\frac{Q(s,b)}{t_0}}} \quad (4.29)$$

The temperature parameter t_0 plays the same role as the ϵ parameter in a ϵ -greedy policy. When t_0 tends to zero, the probability of the greedy action tends to 1, whereas when t_0 tends to infinity, the probability distribution converges towards a uniform distribution. An example of softmax policy is given in Fig.4.2.

It can be demonstrated that improving a policy by making it ϵ -greedy or softmax with respect to Q does improve the policy at each state. Besides, using one of these two kinds of policy families ensures that $\forall s, a, \pi(s, a) > 0$. On top of that, if these policies tends to greedy policies, then they satisfy the property of being *greedy in the limit of infinite exploration* (GLIE), which in case of MC can be proved to be a criterion of convergence [Sutton 1998]. We will not go into the details of the convergence demonstration, but its existence is indeed a very important result that we have to keep in mind.

Before talking about the last class of algorithm (RL), let's notice once more that solving a MDP with Monte-Carlo is already a way to learn from interaction. These methods already raise the problem of the exploration-exploitation trade-off, sampling, infinite exploration and policy improvement that we will retrieve in Reinforcement Learning, coupled with DP ideas.

4.2.4 Reinforcement Learning

Now that we have a clear understanding of what a MDP is, and how to solve it with DP or MC, we can naturally introduce a last class of algorithms that somewhat bridges the gap between the two methods mentioned above. Remember that the strength of DP is to rely on a one-step prediction based on the Bellman equation, and that the strength of MC is sampling. Reinforcement Learning combines both at the same time.

Temporal Differences

The key idea is illustrated by an algorithm called Temporal-Differences (TD) to estimate the quality of the policy π by computing iteratively the V^π . Like DP, TD updates the estimates of $V(s)$ based on the estimates of the next states $V(s')$, but unlike DP, it does not use all the next states for this update. Like MC, TD samples one action at each time step, but unlike MC, it does not wait until the end of the process to update its estimate of $V(s)$. The update equation of TD is:

$$V_{k+1}(s) = V_k(s) + \alpha_k [r + \gamma V_k(s') - V_k(s)] \quad (4.30)$$

where r is the observed reward obtained after playing action a in state s . $r + \gamma V_k(s')$ is the observed approximation of $V^\pi(s)$ and $V_k(s)$ is its current estimate.

In this update equation appears a learning rate $\alpha_k \in [0, 1]$, which may vary with time. With comparison with the MC update presented in algorithm 4.2 the learning rate was $1/n$, with n the

Algorithm 4.3 SARSA

$\forall s, a \quad Q(s, a) = 0$
loop
 $s \leftarrow$ first state of the episode
 $a \leftarrow$ sample from $\pi(s, \cdot)$
repeat
 Take action a , observe s', r
 $a' \leftarrow$ sample from $\pi(s', \cdot)$
 $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$
 $\pi(s, \cdot) \leftarrow$ ϵ -greedy with respect to $Q(s, \cdot)$
 $s \leftarrow s'$
 $a \leftarrow a'$
until $s =$ end of the episode
end loop

number of visits of a state-action pair. We will meet again this general update scheme, where the old estimate of V (or Q) is moved toward the current observation with a rate α . The stochastic approximation theory has proved that such a sequence $V_k(s)$ does converge according to such an update scheme if the learning rate sequence α_k satisfies the following conditions:

$$\sum_{k=1}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty \quad (4.31)$$

This is especially the case for the MC learning rate. However, a constant learning rate is sometimes used when the environment itself is evolving with time. In our case, as we will see soon, the environment is completely static.

This DP idea of updating V right after getting the immediate reward is indeed very natural. There exist many learning tasks for which waiting till the end of the process is not the most efficient way to update our estimate, since the following decisions may not be all related to this very specific one. Therefore, depending on the problem itself, MC method will need more iterations to converge, especially when dealing with local decisions that do not affect the whole process.

TD estimates V^π . We can combine it with policy improvement to iteratively compute the optimal policy. However, a model of the environment dynamics is not always available and therefore, working on the action-value function Q is often preferred. The two most famous algorithms to estimate the optimal policy π^* that we are going to introduce now are SARSA and Q-learning.

SARSA

SARSA is somehow the natural extension of TD idea to the estimation of the optimal Q-function. This method is given in algorithm 4.3. The name of SARSA comes from the way the algorithm proceeds. In state s , take action a , observe a reward r , and new state s' . Sample here an action a'

Algorithm 4.4 Q-Learning

$\forall s, a \quad Q(s, a) = 0$
loop
 $s \leftarrow$ first state of the episode
repeat
 $a \leftarrow$ sample from $\pi(s, \cdot)$
 Take action a , observe s', r
 $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
 $\pi(s, \cdot) \leftarrow \epsilon$ -greedy with respect to $Q(s, \cdot)$
 $s \leftarrow s'$
until $s =$ end of the episode
end loop

according to the current policy, and use the estimate of $Q(s', a')$ to update the estimate of $Q(s, a)$. Therefore the sequence of states, actions and reward is s, a, r, s', a' : SARSA. As in MC, the process is repeated over many episodes: we have to play the game many times before mastering it!

SARSA converges with probability 1 towards Q^* under the stochastic approximation conditions, and using a policy greedy in the limit of infinite exploration (GLIE). Q-learning is another algorithm proposed by Watkins in 1989 [Watkins 1989] proved to be usually faster in practice. It was also proved to converge with probability 1 towards Q^* under the same conditions by Watkins and Dayan [Watkins 1992].

Q-Learning

The idea of Q-learning is indeed closer to the Bellman optimality equation. The idea is not to update with the estimate of a sampled state-action pair (s', a') , but based on the currently best next estimate $\max_{a'} Q(s', a')$. An action is still sampled to go from s to s' , but the sampled action a' is not necessarily the one used for the update. Therefore it is often said that Q-learning is estimating a policy while following another one. More formally, the Q-learning update is:

$$Q_{k+1}(s, a) = Q_k(s, a) + \alpha_k \left[r + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right] \quad (4.32)$$

For convenience, the Q-learning algorithm is given in pseudo code in algorithm 4.4. Please note that by setting $\alpha = 1$, we obtain a update equivalent to the value iteration one in DP when the environment is deterministic (see algorithm 4.1). However, the difference between Q-learning with unit learning rate and value iteration in a deterministic environment is that Q-learning still *samples*, whereas value iteration *sweeps*. Q-learning samples trajectories in the state space according to the decision process while value iteration keeps traversing the whole state space.

We introduced RL methods as lying somewhere in between DP and MC. We discussed here the connections between these three classes of algorithm, and the strong similarities between MC and

RL. Indeed, this link can be reinforced by extending TD, SARSA or Q-learning to sampling several states ahead rather than only one. These methods called $TD(\lambda)$, $SARSA(\lambda)$ and $Q-learning(\lambda)$ are not detailed any further in this thesis.

4.2.5 Semi Markov Decision Processes

Before entering the details of the parsing algorithm, we need to introduce the concept of semi Markov Decision Process (SMDP) [Barto 2003] since they are particularly well suited to handle hierarchies of tasks rather than flat sequences of decisions.

The idea of SMDPs is that some actions may take more time than others. As a consequence, we add a random variable τ called the *waiting time* and that indicates how much the agent has to wait to complete an action. Here, we merely consider discrete-time semi Markov Decision Processes. In this framework, the actions are taken and completed at regular time steps, all multiples of a implicit unit time step. The transition probabilities now take into account the next state s' and the waiting time τ : $P(s', \tau | s, a)$.

The Bellman optimality equation for Q naturally becomes:

$$Q^*(s, a) = R_{ss'}^a + \sum_{s', \tau} \gamma^\tau P(s', \tau | s, a) \max_{a'} Q^*(s', a') \quad (4.33)$$

DP algorithms, such as value iteration, can be adapted to SMDPs using equation 4.33 instead of 4.18 as an update rule. Moreover, methods based on direct interaction between the agent and its environment such as MC or RL also cover the case of SMDPs. For instance the Q-learning update of the estimate of Q^* becomes:

$$Q_{k+1}(s, a) = Q_k(s, a) + \alpha_k \left[\sum_{i=0}^{\tau-1} \gamma^i r_{t+1+i} + \gamma^\tau \max_{a'} Q_k(s', a') - Q_k(s, a) \right] \quad (4.34)$$

Obviously, we retrieve the Q-learning update in the case where each action a has a waiting time $\tau = 1$. The main advantage of SMDP is that they can naturally model hierarchies of tasks. Some tasks are recursively made of subtasks. The reward associated to a task is equal to the sum of the rewards of its subtasks. This will be particularly convenient while modeling the shape grammar parsing decision process. Moreover, the algorithms available in standard Reinforcement Learning are still efficient in SMDP. Please note that these extensions of Q-learning or SARSA are not the only methods available in Hierarchical Reinforcement Learning. Indeed, a lot of work have been proposed to take advantage of hierarchies of tasks [Barto 2003], the most famous one being Dietrich's MAXQ algorithm [Dietterich 2000].

This section has presented the framework of Markov Decision Process and different families of solutions. In the remainder of this chapter, we will explain how the shape grammar parsing can be reformulated in the presented framework and we will then discuss the best way to optimize it based on its specific properties.

4.3 Parsing Split Grammars

Having a better understanding of the Markov Decision Processes and the 2D binary split grammars (BSGs), we are about to reformulate the problem of parsing a BSG within this framework. The change of viewpoint is the key that opens the door to the classes of algorithms presented in section 4.2 and that were so far barely explored in Computer Vision. We will first properly define the grammar parsing problem. Then, we will explain how to formulate the grammar derivation within this new MDP framework first in the 1D case, and then in the 2D one, paying attention on the choices of states and their impacts on the optimization of the decision process.

4.3.1 The Parsing Problem

Let's first define what the objective of the parsing problem is. The inputs are:

1. \mathcal{I} : a rectified image of a facade. The image has dimensions $W \times H$. In this chapter, we assume artificial data that is to say that for each pixel x of $\mathcal{I}(x)$ is a terminal symbol c of the BSG.
2. G : a Binary Split Grammar (context-free).
3. $m(x, c)$: a *merit* function that specifies for each type of terminal symbol $c \in \mathcal{T}$, the appropriateness of pixel $x \in \mathcal{I}$ being labeled as c . The higher is the merit the more likely the pixel x belongs to class c . Working with artificial data, this merit function is straightforward: $m(x, c) = \delta(\mathcal{I}(x), c)$ It is 1 if the pixel x represents the symbol c and 0 if it corresponds to any other symbol. We will define in chapter 5 other merit functions to parse real data. The definition of the merit function does not change the mechanism of the parsing algorithm.

Remember that every sentence C of the language $L(G)$ is a semantic segmentation of \mathcal{I} , therefore the parsing goal is to find the sentence that maximizes the sum of the merits:

$$C^* = \arg \max_{C \in L(G)} \sum_{x \in \mathcal{I}} m(x, C(x)) \quad (4.35)$$

The question is now: how to optimize this parsing function? In order to solve it, we are now going to reformulate the derivation process as a Semi Markov Decision Process. Before tackling the complete problem, we suggest first to have a look at a simpler problem in which the binary split grammar works on a single dimension. Having understood and solved this simple problem, we will be able to tackle the complete 2D parsing of context-free binary split grammars.

Note that formulating a PCFG parsing as an MDP is not by itself a contribution of this thesis, since it was already detailed in [Neu 2009]. However, the application to shape grammar parsing and the proposed decision process are claimed to be a contribution of this doctoral work.



Figure 4.3: Example of image to be parsed with a 1D split grammar.

4.3.2 1D parsing

Consider that we have a 2D image that represents an irregular sequence of black and white rectangles such as shown in Fig.4.3. This image can be seen as a floor, where the wall is black and the windows are white. Therefore, it can easily be modeled by a 1D grammar, such as the one given in table 2.1. We suppose that the axiom of the grammar is $\text{floorWa}(0, 0, W, H)$.

While deriving the grammar, the derivation process chooses first a rule that puts a wall of a certain width, then a window, and so on and so forth until the image has been entirely segmented by an alternate sequence of walls and windows. Formulated that way, it seems that we could see the parsing problem as an MDP. An example of episode with the corresponding states, actions and rewards is given in table 4.3.2.

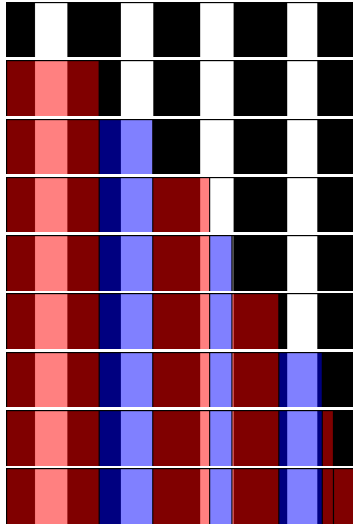
1D parsing as MDP

In order to formulate the 1D parsing problem as an MDP, we have to explicitly specify what the states, the actions, the transition probabilities and the rewards are and check that the sequence of decisions satisfies the Markov property (see equation 4.4).

Agent = derivation process As mentioned before, at each time step of the derivation, a node (atomic shape) is processed. We can imagine that this process is controlled by a learning agent that decides how big should be the next wall or the next window, by choosing a rule in the grammar.

Environment = partial parse tree The agent is interacting with the current segmentation of the image, or similarly, we can say that the agent is interacting with the partial parse tree. This partial parse tree is sequentially built by the agent's decisions. In our case, since the two types of rule create one terminal symbol and one non-terminal symbol, the partial parse tree always has at most a single leaf that corresponds to a non-terminal symbol and that has to be further derived (see Fig.4.4).

States = non-terminal atomic shapes = (symbol, position) A first possibility is to consider that each partial parse tree is a state. Notice that there might be a very large number of partial parse trees. Since the grammar is context-free, the rules only depend on the current non-terminal node of the partial parse tree, and not on the full tree. In other words, the split of an atomic shape (node of the tree) $\text{floorWa}(x, y, w, h)$ does not depend on any other atomic shapes (node) of



s_t	a_t	s_{t+1}	r_{t+1}
(floorWall, 0)	(1,83)	(floorWin, 83)	2095
(floorWin, 83)	(2,48)	(floorWall, 131)	1470
(floorWall, 131)	(1,51)	(floorWin, 182)	2156
(floorWin, 182)	(2,20)	(floorWall, 202)	1078
(floorWall, 202)	(1,43)	(floorWin, 245)	2107
(floorWin, 245)	(2,38)	(floorWall, 283)	1372
(floorWall, 283)	(1,10)	(floorWin, 293)	490
(floorWin, 293)	(2,30)	(end, 311)	882
		return	11650

Table 4.1: Step-by-step episode of the 1D parsing. At each time step, the agent chooses an action that adds a terminal symbols on the current segmentation, and from which a reward is computed. The goal of the agent is to learn the best sequence of decisions so as to maximize the sum of the rewards (return). The walls of the image are black, and the one of the segmentation are red. The windows of the input image are white and blue in the segmentation. We superimpose here the partial segmentation with the original image. In this example, the complete return is 11650 whereas the optimal return is 15239. This optimal solution can be obtained with value iteration or Q-learning.

the partial parse tree. The agent does not need to know anything else while taking his decision of how to split an atomic shape. As a consequence, we define the states as *the non-terminal atomic shapes*. In this specific case, we are dealing with a 1D grammar; therefore the parameters y and h of the rectangles do not vary. Furthermore, based on the definition of the two kinds of split rules, the width of the rectangles of the non-terminal atomic shape w , is directly related to its position x , since $w = W - x$ (W is the width of the image). As a consequence, the state s can be represented as $(floorWa, x)$ or $(floorWin, x)$, since there exist only two kinds of non-terminal symbols in this grammar.

Actions = rules = (id, width) Obviously, in states $s = (floorWin, x)$ the possible actions are the rules which LHS is $floorWin$. In states $s = (floorWa, x)$, the possible actions are the ones with $floorWa$ as LHS of the corresponding rules. Consistently with the way we denoted the rules in 2.1, we will denote the actions $a = (id, w)$, where id is the id of the rule in the grammar and w the associated parameter (the width of the terminal element).

Rewards = sum of merits Remember that we assume to have a merit function that tells us the quality of every pixel for every possible terminal symbol of the grammar. Let's suppose that the agent is in state $s_t = (\text{floorWa}, x)$, in an image of dimensions $W \times H$. It means that currently the atomic shape being processed is $\text{floorWa}(x, 1, W-x, H)$. Then the agent chooses an action $a_t = (1, w)$. The consequence of this action is the creation of two atomic shapes: a terminal one $\text{wall}(x, 1, w, H)$ and a non-terminal one $\text{floorWin}(x+w, 1, W-(x+w), H)$. This last non-terminal shape constitutes the new state: $s_{t+1} = (\text{floorWin}, x+w)$. This transition is completely deterministic. The reward received by the agent only depends on this choice and is computed based on emerging terminal shapes through the merit function m . Since the terminal shape added here is a wall, we have:

$$r_{t+1} = \sum_{i=x}^{x+w} \sum_{j=1}^h m((i, j), \text{wall}) \quad (4.36)$$

Returns = partial parsing function The good news is that the returns that the agent tries to optimize are exactly coinciding with the parsing function of equation 4.35. Let x_t, x_{t+1}, \dots, x_T be the consecutive positions of the agent from time step t to the end, and c_{t+1}, \dots, c_T the consecutive terminal symbols added to the segmentation (alternatively walls and windows) then the return R_t is:

$$R_t = \sum_{k=t+1}^T r_k = \sum_{k=t+1}^T \sum_{x=x_{k-1}}^{x_k} \sum_{y=1}^H m((x, y), c_k) \quad (4.37)$$

We can denote $C(x, y) \in \mathcal{T}$ the image of terminal symbols sequentially built by the agent. Since the first state s_0 is $(\text{floorWall}, 0)$ and that the last state s_T is (end, W) . Then the complete return of the decision process is:

$$R_0 = \sum_{x=1}^W \sum_{y=1}^H m((x, y), C(x, y)) \quad (4.38)$$

Thus, solving the parsing problem defined in equation 4.35 is the same thing as finding the best sequence of actions to take from the axiom to the end.

Markov Property The Markov property is satisfied by this decision process since the grammar is context-free. The next state (atomic shape) is completely determined by the current state (LHS of the rule) and the chosen action (the rule itself).

This reformulation is natural. The agent sequentially chooses the rules to apply, changes the current segmentation, and receives a reward corresponding to this local modification. Since the parse tree is binary, and that each node has at most one child which is non-terminal, the parse tree

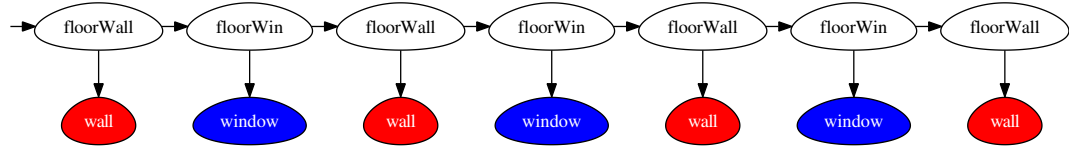


Figure 4.4: The parse tree is a decision process. The agent moves from left to right, the states are the white nodes. At each white node, the agent takes a decision. By doing so, he creates two nodes: the color node below is a terminal node which provides a reward. The white node on the right in non-terminal and constitutes the agent’s next state.

can be seen as a sequential process (see Fig.4.4). The decisions are taken at each white node. The consequence of the agent’s decision sends him to the next white node on the right, and the reward he receives is computed based on the colored-node below. In this 1D example, the agent simply chooses how many consecutive pixels he labels as window (blue) or wall (red). Optimizing the MDP is strictly equivalent to optimizing the parsing problem due to the definition of the rewards. Note that the complete return counts the merit of each pixel exactly once. Now that we have expressed the parsing problem as a MDP, let’s study how to solve it.

Solving the 1D parsing

Before reviewing the different solutions, note that the action that gives the biggest immediate reward is usually not the best action to play. For that matter, remember that the merit function is 1 when the agent is right, and 0 when he is wrong. The reward functions are given in Fig.4.5. These curves show how much the agent gets if he is in position $x = 0$ and puts a window (blue) or a wall (red) of dimension w . Starting from the axiom (`floorWall`, $x=0$), the action that gives the biggest immediate reward is the one that put the largest wall, since the rewards are non-decreasing. By doing so, the agent does not optimize the long term reward, since he would get more by alternating walls and windows, avoiding the constant parts of the reward functions.

As mentioned in section 4.2, there are three main classes of methods to solve such an MDP when the model of the environment is known. The idea is always to estimate the optimal action-value function Q^* . Indeed, this function has the good property that for each state s the greedy action a^* such that $a^* = \arg \max_a Q(s, a)$ is also the action that guarantees the biggest long-term reward. Therefore, if Q^* is known, the optimal sequence of rules is obtained by starting from the axiom, and keeping choosing the greedy action with respect to Q^* .

In the 1D case, the number of states is quite small. In our toy example, the image is 311 pixels wide and 49 pixels high. There are at most $311 \times 2 = 622$ states. Knowing that the number of actions per state is quite small as well (~ 70), it is possible to solve this MDP using a DP algorithm such as value iteration (see algorithm 4.1). For comparison’s sake, we also provide a Q-learning solution. We do not show the visual segmentation obtained by both methods since they coincide with the perfect segmentation we would expect.

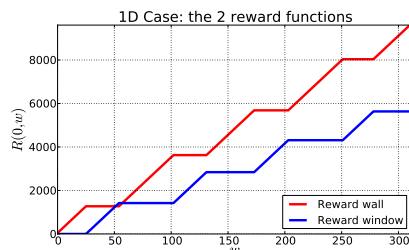


Figure 4.5: Reward functions in the 1D case. We suppose to start from position $x = 0$. The red curve gives the reward that the agent gets if he decides to apply a rule that puts a wall of size w . The blue curve is the equivalent for choosing a window of size w . The reward function is non-decreasing, and has constant parts, since putting a wall in a window region does not bring any reward and vice-versa. Note that optimizing the immediate reward does not optimize the long-term one.

DP Solution As mentioned earlier, DP guarantees the optimality of the solution. Using a grammar with 162 rules, divided into two kinds as in table 2.1, with 71 rules to add a window and 91 rules to add a wall, we obtain in practice 594 states, and the value iteration algorithm converges in 7 sweeps over the state spaces. In term of speed performance, the result is obtained in around 12 seconds on a standard laptop. For comparison's sake, we computed the Q-function and show it in Fig.4.6 for the different position of the symbol `floorWin`, and different possible actions. The redder the value of $Q(s, a)$, the higher. Of course the values are decreasing while the position is increasing. As we get closer to the end of the process the expectation of accumulated rewards decreases. However, we can also notice some important peaks into the Q-function, corresponding to particularly interesting moves for the agent.

RL Solution Unlike DP, Q-learning does not guarantee the optimality, but guarantees to converge toward the optimal solution. Using the same grammar, we ran a Q-learning agent for $T = 10000$ episodes, using a ϵ -greedy policy, exponentially decreasing from 1.0 to 0.01, and with a learning rate decreasing exponentially from 1 to 0.01. The run-time is about the same as the one with DP. The learned Q-function is given in Fig.4.6 as well. We can notice a first important difference. The Q-function is not completely filled, and not uniformly as well. The Q-agent explores the state-space, but gives more credit to the reward-wise interesting regions. It is clear both on the full image of Q as well as on two cuts. On the first one (lower left in Fig.4.6), we are in a region of the state space that pays-off. The exact DP solution in dashed red is well approximated by the Q-learning one represented in solid blue. On the contrary, in the middle of a poorly interesting region (lower right) the estimate of Q is far from its exact value. The agent has realized early enough that this part of the state space was not worth exploring deeply, and preferred focusing on more promising ones.

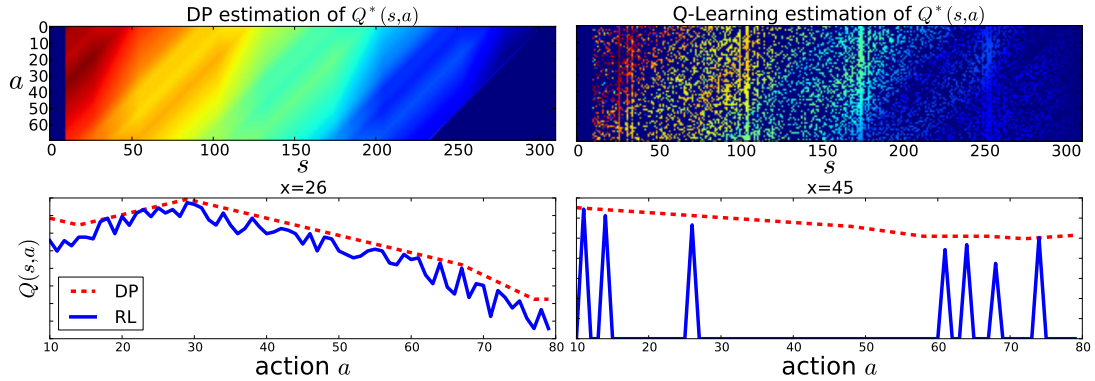


Figure 4.6: Comparison of DP and RL on the 1D parsing. On the first row, we represent the estimated Q^* function for the symbol `floorWin`. On the left is the Q function estimated by DP and on the right the Q function estimated by Q-learning. For each state-action pair s, a corresponds a value represented by the color of the pixel $Q(s, a)$. Deep blue means 0, the redder the higher. On the second row, we represent two vertical cuts of the same functions. One the left, the Q functions for $x = 26$ and on the right for $x = 45$. The dashed red line represents the exact estimation by DP and the blue line the approximate estimation by Q-learning.

Conclusion on 1D parsing This first 1D example tells us a lot about the parsing problem. First we see that the parsing problem can be exactly expressed as a MDP. Then, we see that in this specific case, DP is indeed the best option since it exactly solves the problem in the same amount of time. However, we already see that DP is too brute force. It solves the parsing problem and indeed much more than the problem. Starting in any unpromising state, DP leads us to the best possible solution. However, what interests us is not to start from anywhere in the state space, but to start from the axiom, and therefore the complete knowledge of the best solution at every state might be a luxury that we will not be able to afford in 2D.

4.3.3 2D Parsing

We have seen that the 1D parsing problem can easily be formulated as an MDP. Now the question that remains is how to perform the 2D parsing? Imagine that we have as input the image a stack of floors such as the one in Fig.4.7. The idea is again to express the parsing problem as a Markov Decision Process. Since the procedural mechanisms are the same in 1D, 2D or 3D, we can still consider an agent, responsible for deriving the shape grammar. He still follows a DFS scheme and faces a sequence of decisions. As a consequence, the decisions are still the rules: the actions of the 2D decision process remain the same. Furthermore, the rewards are still computed based on the merit functions that quantify the quality of the terminal symbols. However, in the 2D case, some rules are not placing any terminal symbols but rather non terminal symbols only. The agent now

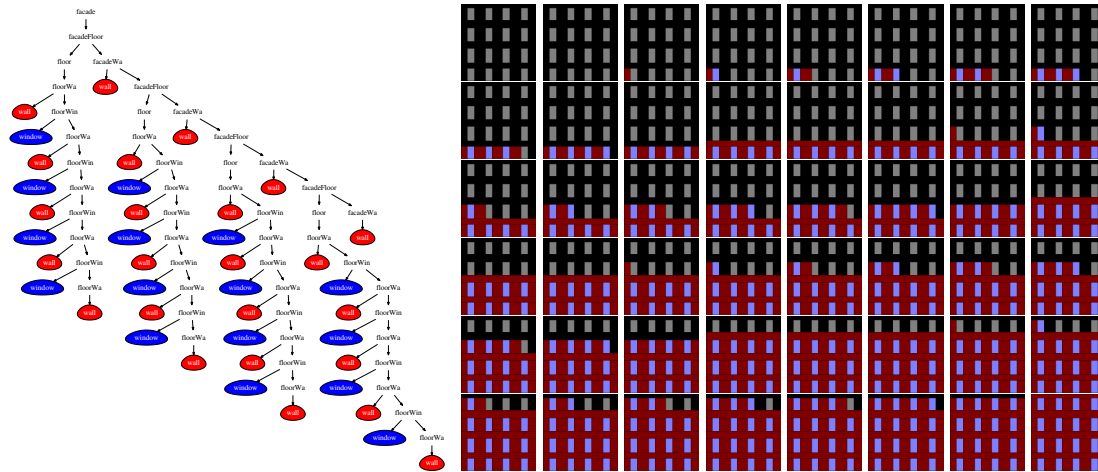


Figure 4.7: A complete derivation in DFS order. This specific derivation is made of 48 decisions. We show the original image blended with the semantic segmentation at each step.

faces a hierarchy of tasks. We will define the states of the decision process and then explain why we have a hierarchy of tasks. To get a better understanding of the decision process in 2D, Fig.4.7 shows all the steps of a derivation and the corresponding parse tree.

A First State Definition

In 1D, the current state of the environment was defined as the left-most non-terminal node of the parse tree. Let's take the same definition: $\text{symbol}(x, y, w, h)$. Exactly as in the 1D case, we can define a MDP that can be solved with Dynamic Programming.

However, the dimensionality of the problem is now way bigger than it used to be, since the state as 4 parameters instead of 1. The state space explodes, and even if it could in some cases fit into memory, the running time would be huge since the agent has to perform several complete sweeps in the state space before converging. For instance, in a simple example with 3 floors and 4 windows, parsing the binary segmentation grammar given in table 2.2 with DP takes almost 2 full days on a standard machine. The relevance of this approach is clearly debatable.

Then, the run-time is not the only problem of this MDP, but also what decisions such an agent would be learning. Since the states are the atomic shapes $\text{symbol}(x, y, w, h)$, two different floors have different vertical positions y and there we allow the agent to take different greedy decisions on different floors. This may not be a problem on artificial data, but could turn out to be one on real data. In all likelihood, if we allow the agent to learn different decisions on two different floors, then it is highly probable that he will learn different decisions, because the information provided by merit m is not necessarily perfect (see section 5.1).

The consequence of such a state definition is that the agent will learn slowly to take inconsistent decisions. If we want the agent to take the same decisions at two different floors, then we should

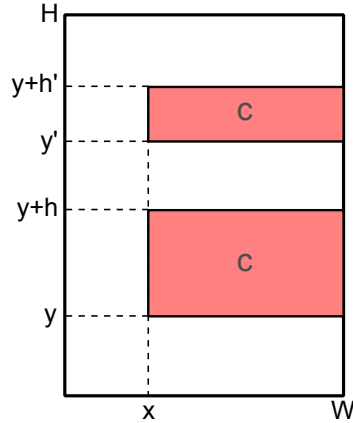


Figure 4.8: State Aggregation. Two different atomic shapes in red $c(x, y, w, h)$ and $c(x, y', w', h')$ are mapped to the same state, since they share the same position x along X (supposed to be the split direction) and the same symbol c , even though they are not the identical.

force the two states to be the same. Thus, we propose to modify the state definition in order to decrease the dimension of the problem and guarantee to have consistent solutions. The solution is closer to the grammar spirit itself.

State Aggregation

Based on the aforementioned limitations, we propose a new state definition:

$$s = (c, x)$$

where the c is still the symbol carried by the left-most non-terminal atomic shape and x is the position of the atomic shape *in the splitting direction*. This concept is illustrated in Fig.4.8. The idea is that while splitting along a direction, the positions along other dimensions are irrelevant. Note that the grammar defines the geometry of a split rule only in the dimension of the split, regardless any other one.

In other words, we *aggregate* the states. Many atomic shapes are mapped to the same state. The first advantage is to reduce the dimension of the state space, which now shares the same order of magnitude as in the 1D case. The second advantage is to ensure consistency along the facade. Let's explain this last statement.

Remember that we aim to estimate a Q^* function. To obtain the optimal parse, at each state s the chosen action is greedy with respect to Q^* : $a^* = \arg \max_a Q^*(s, a)$. Since the state only depends on the position along the split direction, two floors will be split in the same way along X since the states describing the atomic shapes inside the two floors will be identical (independent of the Y direction). This statement extends to more complex grammars.

State aggregation reduces the dimensionality of the state space and ensures consistency. The price to pay to obtain these properties is the stochasticity of the rewards and therefore the impossibility to use Dynamic Programming to solve the decision process.

Stochastic Rewards

The consequence of state aggregation is that the rewards become stochastic. This comes from the difference between the full description of the environment hidden from the agent and the state signal sent to the agent. At each iteration, the environment is still a partial parse tree, made of atomic shapes, and one node is selected in the DFS order to be derived by the agent. However, the agent does not know that he is about to derive the atomic shape $c(x, y, w, h)$, he only knows the incomplete information (c, x) or (c, y) (depending on the split direction).

Let's imagine that the agent splits this atomic shape $c(x, y, w, h)$ along X with action $a = (id, v)$ leading to a terminal atomic shape ν_1 of symbol c' and a non-terminal atomic shape ν_2 . Then the environment still sends a reward r as in the 1D case:

$$r = \sum_{i=x}^{x+v} \sum_{j=y}^{y+h} m((i, j), c') \quad (4.39)$$

Now, let's suppose that at two different iterations the agent visits two nodes of the partial tree $c(x, y, w, h)$ and $c(x, y', w', h')$ that have to be split along X . Then for these two nodes, the state signal sent from the environment to the agent is the same: $s = (c, x)$ (see Fig.4.8). If in both cases the agent takes the same action $a = (id, u)$, then the rewards he gets the first time will be different from the one he gets the second time, since they are not computed on the same regions of the image. From the agent point of view, the rewards are stochastic: applying the same rule in the same state can produce different rewards.

Hierarchical Learning

So far, we have assumed that after each decision, the agent received a reward. When the action creates at least one terminal symbol, this statement holds. However, if the action only creates non-terminal symbols, then it raises the question of evaluating the quality of such an action.

For example, a rule might split a `facade` into a `floor` and the remainder of a `facade`. We observe here, that a `floor` is more a parsing subtask by itself, and therefore we will be able to quantify the quality of this decision only once the `floor` task will be completed. In this decision process, some actions take more time than others.

Note that we do not know in advance when a task will end. Thus, we are facing a semi Markov Decision Process (SMDP). Pay attention that the transition probabilities $p(s', \tau | s, a)$ become here partially stochastic: based on s and a we know for sure what is going to be the next state s' but we do not know when we will reach it. τ is a random variable. Therefore, when the agent is facing a task, he receives at the end of it the sum of all the rewards he got meanwhile as a reward for having completed it. It might be that the task itself is recursively made of subtasks.

More formally, we divide the set \mathcal{N} of non-terminal symbols into two: \mathcal{N}_0 and \mathcal{N}_+ . The symbols in \mathcal{N}_0 are non-terminal symbols that are not subtasks. For instance `floorWall` or `floorWin` such as defined in the 1D case are not subtasks. The second set \mathcal{N}_+ represents the subtasks. For instance the symbols `floor` and `facade` belong to this category. In practice, the distinction between the two types of non-terminal symbols is specified in the definition of the grammar, some symbols are manually specified as subtasks.

Now let's suppose that at time-step t , the agent is deriving the node ν of the parse tree that corresponds to atomic shape $c(x, y, w, h)$. After playing an action a , two atomic shapes ν_1 and ν_2 are added as children of ν . We defined the reward r consequent to action a as the merit of node ν by extending the definition of the pixel-wise merit function to node-wise merit function:

$$m(\nu) = m(c(x, y, w, h)) = \begin{cases} \sum_{i=x}^{x+w} \sum_{j=y}^{y+h} m((i, j), c) & \text{if } c \in \mathcal{T} \\ m(\nu_1) + m(\nu_2) & \text{if } c \in \mathcal{N}_+ \\ 0 & \text{if } c \in \mathcal{N}_0 \end{cases} \quad (4.40)$$

This recursive definition of the merit function is in line with the hierarchy of tasks; some tasks may be composed of other tasks. This hierarchy is also illustrated in Fig.4.7. Unlike in Fig.4.4, where the sequence of states is represented by a chain, the sequence of states in the 2D case is branching. Then, the reward for taking action a in this configuration is simply computed as: $r = m(\nu)$. Note that this definition is consistent with the 1D case.

Applying this definition of the merit at the root ν_{root} of the parse tree, we obtain that the merit of the root node is recursively computed on the leaves of the tree, which are a partition of the input image. Once more, these leaves constitute a semantic segmentation C , and therefore:

$$R_0(C) = m(\nu_{root}) = \sum_{x=1}^{x=W} \sum_{y=1}^{y=H} m((x, y), C(x, y)) \quad (4.41)$$

Optimization strategy

Seeing the decision process as a Semi Markov Decision Process makes the transition probability stochastic and unknown. Aggregating the states make the reward stochastic and unknown. As a consequence, in the 2D case, the use of Dynamic Programming is not possible, since the dynamic of the environment is unknown.

According to section 4.2.2 Monte-Carlo and Reinforcement Learning are the two remaining options. MC methods are not really well adapted to the current method, since the decisions taken by the agent are local. The method was already proved to be quite slow in [Teboul 2010]. As a consequence, Reinforcement Learning and more particularly Q-learning seem to be the best candidate to solve the 2D parsing problem.

Algorithm 4.5 describes the parsing algorithm. The agent keeps building the segmentation (environment) by choosing one rule (action) at a time. However, the agent does not know the exact description of this segmentation, but only has a partial representation (state). By applying a rule,

Algorithm 4.5 Parsing BSG via Q-Learning

```

 $\forall s, a \quad Q(s, a) = 0$ 
 $\forall s, a \quad p(s, a) \sim Uniform$ 
 $\epsilon = \epsilon_0$ 
while  $n < T$  do
  reset environment
   $s \leftarrow (Axiom, 0)$ 
  update  $\epsilon$ 
  repeat
     $a$  sample from  $\pi(s, \cdot)$ 
    Take action  $a$  (recursively if subtask), observe  $s', r$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
     $\pi(s, \cdot) \leftarrow \epsilon$ -greedy with respect to  $Q(s, \cdot)$ 
     $s \leftarrow s'$ 
  until  $s = \text{end of the episode}$ 
end while

```

the agent may create a terminal symbol, a subtask or a non-terminal symbol which is not a subtask. He receives a reward computed on the terminal symbols and gathering all the rewards obtained during the subtasks. After the application of a rule, he gets a reward r and reaches a state s . The action-value function is iteratively learned by Q-learning updates.

Analysis of Convergence in the Artificial Case

According to the theory, the Q-learning algorithm converges towards the optimal policy. Since we have slightly modified the process by aggregating the states, it is still interesting to analyze how the algorithm performs in practice. For that matter, we consider the following test case. The input is a simple binary white/black 2D facade with 3 floors and 4 windows that we want to parse with the binary segmentation grammar presented in table 2.2. We run a $N = 1000$ identical Q-learning agents for $T = 5000$ episodes. The learning rates α of the agents are exponentially going from 1 to 10^{-3} and the exploration rates ϵ is decreasing exponentially from 1 to 10^{-3} . For each agent i and for each episode k , we get a full return R_i^k (the value of the parse tried by the agent), as well as a greedy return G_i^k at episode k . It represents what the agent gets while following the greedy policy (learned from Q) after k episodes. For a given agent i , the trajectories R_i^k and G_i^k are quite noisy, as we can see in Fig.4.9(a). To get rid of the intrinsic stochasticity of the agent's exploration, we average the N trajectories so as to obtain the curves in Fig.4.9(b).

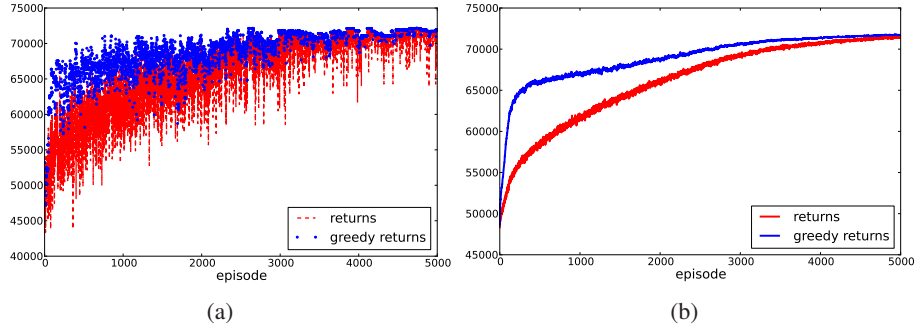


Figure 4.9: Convergence of the RL parsing algorithm. (a) The trajectories of returns and greedy returns for a single agent. Due to the ϵ -greedy policy, the two trajectories are very noisy. However, in (b) we can see the learning trend by averaging among 1000 identical agents. The learning curves are clearly increasing.

$$\begin{aligned}\bar{R}^k &= \frac{1}{N} \sum_{i=1}^N R_i^k \\ \bar{G}^k &= \frac{1}{N} \sum_{i=1}^N G_i^k\end{aligned}\tag{4.42}$$

We can see in Fig.4.9 that the two curves converge towards the same limit. This is no surprise since the exploration rate ϵ goes to zero, and therefore the agent is almost not exploring at the end. However, the convergence of the parsing algorithm is illustrated by Fig.4.9, since the average curves are increasing. This is an important fact. The more the agent parses the images, the better is the parse. The exact convergence towards the optimal parse depends on the number of episodes, the learning rate and the exploration rate, which are the only 3 parameters of the algorithm.

4.4 Going Further with Reinforcement Learning

We have explained so far how to parse a BSG using Reinforcement Learning, formulating the parsing problem as a semi Markov Decision Process that is optimized using the Q-learning algorithm. However, reformulating the problem gives access to more optimization tools than what was exploited so far. In this section, we explain how to boost the convergence of the Q-learning algorithm using bottom-up cues, how function approximation could be taken into account, as well as model selection, compactness and user-defined constraints.

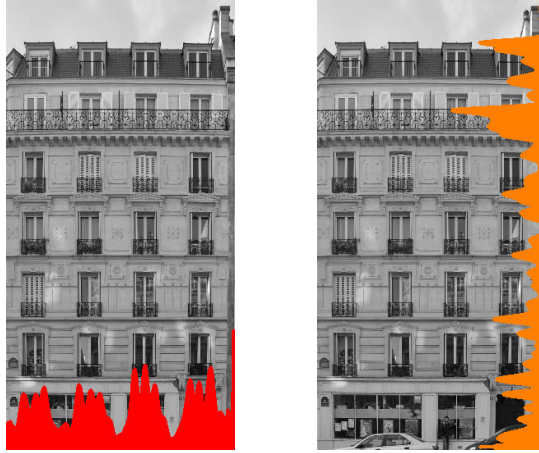


Figure 4.10: Representation of the h_X and h_Y functions on a real image. They represent the cumulative gradients in both directions.

4.4.1 Data-Driven Policy

Following the idea of Data Driven Markov Chain Monte-Carlo presented in [Zhu 2000], we can drive the rule selection process using bottom-up cues. The idea is quite simple: the border between two semantic regions should match the strong gradients along the split direction in the image. Therefore, we propose here to change the ϵ -greedy policy defined in section 4.2.3 so as to take into account the image data.

Remember that a policy π is defined by $\pi(s, a) = p(a|s)$, and that the state s is defined by a 2-tuple $(symbol, x)$, where x is the position along the split direction. As a consequence, while splitting a labeled rectangle along X starting a position x , we should give more credit to action $a = (id, w)$ such as the positions $x + w$ correspond of edges of the image (strong gradients), since it corresponds to the border between two labeled rectangles.

Similarly to [Lee 2004], we build two signals, representing the (smoothed) cumulative gradients in both directions. An example of these two functions is shown in Fig.4.10. They are illustrated on real data, to give a better understanding of their interests:

$$\begin{aligned} \forall i \leq W, \quad h_X(i) &= G_\sigma * \sum_{j=1}^H \|\nabla_x \mathcal{I}(i, j)\| \\ \forall j \leq H, \quad h_Y(j) &= G_\sigma * \sum_{i=1}^W \|\nabla_y \mathcal{I}(i, j)\| \end{aligned} \tag{4.43}$$

Where G_σ is a Gaussian Kernel that smooths the signal. Note that σ can potentially be 0 in

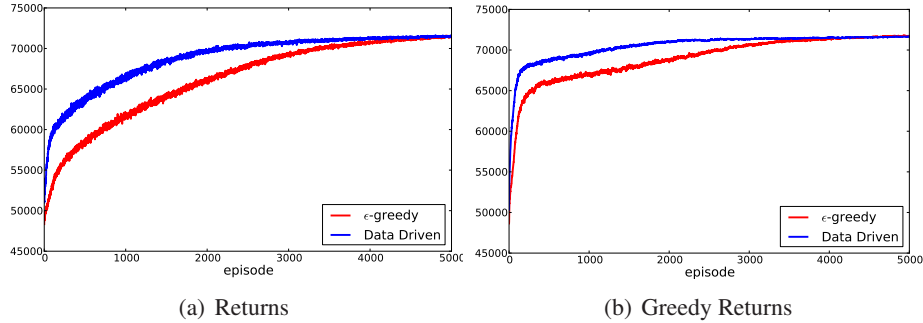


Figure 4.11: Speed-up with data-driven exploration..

case we do not want any blur. These two functions globally indicate where the strong gradients in both directions are, by summing them up all along the image (see Fig.4.10). If in state $s = (c, x)$ we have actions a_1, \dots, a_n splitting along X with parameters w_1, \dots, w_n , and if the action a^* is the current greedy action, then we define the data-driven policy as:

$$\pi(s, a) = (1 - \epsilon)\delta_{a^*}(a) + \epsilon \frac{e^{h_X(x+w)}}{\sum_i e^{h_X(x+w_i)}} \quad (4.44)$$

This equation is valid when s is to be split along X . Obviously, a similar vertical version is straightforward to obtain. Pay attention that the probability to pick up an action that leads to a position with strong gradients is higher than if it leads to weak gradients. Besides, this policy still satisfies the GLIE criterion: as the iteration goes, ϵ goes to zero and the data-driven policy tends to a greedy policy, while keeping non-zeros all the probabilities $\pi(s, a)$. As a consequence, introducing such a policy only guides the exploration but does not compromise the convergence of the Q-learning algorithm.

Experimental Validation

Fig.4.11 shows the difference between using data-driven exploration and standard ϵ -greedy exploration on a simple artificial binary facade, such as the one used in Fig.4.9. In order to be fair, we use the same settings in both sets of experiments, and a Gaussian kernel of non-zero variance ($\sigma = 5$). We observe that the blue curves are always above the red curves, both in the greedy returns as well as the experienced returns. Not surprisingly, data-driven exploration enables the agent to converge faster towards the optimal solution.

Quantitatively, we measure on Fig.4.11 the number of iterations needed to reach 90% of the final return (≈ 72000) with and without data-driven exploration. The agents using a data-driven policy reach this level after 693 iterations in average, whereas they reach the same level after 1761 iterations in average without image-driven cues. Therefore, we can assess that data-driven exploration is 2.5 times faster than ϵ -greedy exploration to reach 90% of the optimal return.

4.4.2 Approximate Reinforcement Learning

One very attractive property of Reinforcement Learning is the support of function approximation. Rather than representing the Q-function as a table with one entry $Q(s, a)$ for each couple state-action (s, a) , the idea is to represent $Q(s, a)$ by some kind of function approximation. One way of doing so is to represent the function as a linear combination of some basis functions ϕ_i :

$$Q(s, a) = \sum_{i=1}^M w_i \phi_i(s, a) = w^T \phi(s, a) \quad (4.45)$$

Note that the basis of functions $\phi = (\phi_1, \dots, \phi_M)^T$ is fixed. As a consequence, the values of Q are ruled only by the weight vector $w = (w_1, \dots, w_M)^T \in \mathbb{R}^M$. This holds for any number of states and actions. Thus, we replace learning the values of Q by learning the weights w . Function approximation also enables us to deal with continuous states and actions, which is quite natural in Computer Vision. Provided that we can still apply RL methods efficiently, function approximation as two main advantages:

1. Reduce the size of the state space, and allow continuous valued states and actions.
2. Better model the intrinsic geometry of the problem.

The first statement is straightforward. To understand the second one, let's have a look at Fig.4.6. We observe that the Q-function learned with Q-learning is very noisy, while we expect it to be much smoother (continuous piece-wise linear in this case). Using function approximation, the idea is to update several values of Q at a time, rather than only one. Since the Q-function is always a linear combination of fixed basis functions, at any iteration the Q-function remains smooth. Please note that using a linear combination of basis function is not the only option for function approximation in RL. For instance, Neural Networks are often used instead of linear models, for their robust and vast approximation capabilities. In this work though, we will restrict the scope of function approximation to linear models. Let's now discuss about how Reinforcement Learning can deal with it.

Stochastic Gradient-Descent

Let's first suppose that we know the target function Q^π that we want to approximate. Then, in function approximation, the goal is to find the best approximation to this target function. *Best* approximation means optimal with respect to a given criterion, usually the mean squared error. Therefore we aim at approximating $Q^\pi(s, a)$ for which we assume to have N samples $q = (q_1, \dots, q_N)^T$ corresponding to N state-actions $((s_1, a_1), \dots, (s_N, a_N))$. Using an linear approximation $Q = w^T \phi$ of dimension M , the means squared error (MSE) is defined by:

$$MSE(w) = \sum_{n=1}^N (q_n - Q(s_n, a_n))^2 = \sum_{n=1}^N (q_n - w^T \phi(s_n, a_n))^2 \quad (4.46)$$

Therefore, approximating using a linear model is indeed exactly a problem of linear regression. Following the principle of the minimum, the gradient of MSE should be 0 at the minimum w^* . This leads to the very classic close form:

$$w_{MLS} = (\Phi^T \Phi)^{-1} \Phi^T q \quad (4.47)$$

where $\Phi \in \mathbb{R}^{N \times M} = (\phi_i(s_j, a_j))_{i,j}$ is called the design matrix, and $(\Phi^T \Phi)^{-1} \Phi^T$ is called the Moore-Penrose pseudo inverse of Φ [Bishop 2006]. However, in the Reinforcement Learning problem, we do not have access to N observations of Q^π ; they come sequentially. However, it is still possible to minimize the mean squared error, through *Stochastic Gradient Descent*. The idea is very simple, starting from a initial w_0 , we iteratively observe the observation $q_t = Q^\pi(s_t, a_t)$, and take into account only the part of the mean squared error involving q_t , by moving the current estimate w_t towards q_t in the direction of the incomplete gradient:

$$\begin{aligned} w_t &= w_t - \frac{1}{2} \alpha \nabla_{w_t} [(Q^\pi(s_t, a_t) - Q_t(s_t, a_t))^2] \\ &= w_t + \alpha [Q^\pi(s_t, a_t) - Q_t(s_t, a_t)] \nabla_{w_t} (Q_t(s_t, a_t)) \\ &= w_t + \alpha [Q^\pi(s_t, a_t) - w_t^T \phi(s_t, a_t)] \phi(s_t, a_t) \end{aligned} \quad (4.48)$$

Knowing that $Q_t = w_t^T \phi$, its gradient with respect to w_t is simply $\nabla_{w_t} (Q_t)(s, a) = \phi(s, a)$. As long as Q^π is known, we can iteratively build an approximation of it through stochastic gradient descent. However, in Reinforcement Learning, the target function Q^π itself is not known in advance, since we aim at estimating it from experience. Therefore, in this specific case, our goal is to build step by step an approximation of a function for which we discover one stochastic sample at a time. In the same spirit as in tabular RL, such an approximation is still possible, replacing the target observation $Q^\pi(s_t, a_t)$ by its estimation. Using Q-learning, the estimation remains the same as before: $Q^\pi(s_t, a_t) \approx r_{t+1} + \gamma \max_{a'} (Q(s_{t+1}, a'))$. Since we consider only the undiscounted case $\gamma = 1$, the update equation of the linear weights w_t after observing s_t, a_t and r_{t+1} boils down to:

$$w_{t+1} = w_t + \alpha \left[r_{t+1} + \max_{a'} Q(s_{t+1}, a') - Q_t(s_t, a_t) \right] \Phi \quad (4.49)$$

Based on this gradient-descent update, we can sequentially learn Q^π . In the Generalized Policy iteration framework, this estimation of π is coupled with a step of policy improvement. Therefore, we can easily derive an approximate Q-learning algorithm (see algorithm 4.6).

Special Cases

Some specific bases are actually related to other concepts we introduced before. For instance, using the Dirac basis or its natural extension, the indicator functions, correspond to either tabular Q-learning or state aggregation, making the whole framework consistent with or without approximations.

Algorithm 4.6 Stochastic Gradient Descent Q-Learning

```

 $w \leftarrow (0, \dots, 0)^T$ 
loop
   $s \leftarrow$  first state of the episode
  repeat
     $a$  sample from  $\pi(s, \cdot)$ 
    Take action  $a$ , observe  $s', r$ 
     $w \leftarrow w + \alpha [r + \gamma \max_{a'} w_t \Phi(s', a') - w_t \Phi(s, a)]$ 
     $\pi(s, \cdot) \leftarrow$   $\epsilon$ -greedy with respect to  $Q(s, \cdot) = w^T \Phi(s, \cdot)$ 
     $s \leftarrow s'$ 
  until  $s =$  end of the episode
end loop

```

Dirac Basis The stochastic gradient descent version of Q-learning is indeed consistent with the tabular case. If the states and actions are finite, then we can take a finite basis of Dirac function to express any discrete signal Q :

$$Q(s, a) = \sum_{s', a'} w_{s', a'} \delta_{s', a'}(s, a) \quad (4.50)$$

Substituting equation 4.50 in equation 4.49 is indeed equivalent to the Q-learning update equation 4.32. Extending Dirac function to indicators function, it is also possible to reformulate state aggregation as a function approximation.

Indicator functions Let's take the example of state aggregation given in the section 4.3.3. We aggregate the all the states $s = (id, x, y, w, h)$ to $\tilde{s} = (id, x)$ through the indicator function:

$$\mathbf{1}_{id, x}(s, a) = \begin{cases} 1 & \text{if } s = (id, x, \dots) \\ 0 & \text{otherwise} \end{cases} \quad (4.51)$$

Then, substituting in equation 4.49 leads to the presented algorithm for 2D parsing. As a consequence, function approximation is genuinely supported by the Reinforcement Learning framework. This extension makes it much more powerful than Dynamic Programming and applicable to a larger set of problems. Let's now discuss one possible way to apply function approximation to our parsing problem, using radial basis functions as an approximation basis.

Radial Basis Function Networks

One very popular approximation basis is called Radial Basis Function Network (RBFN). In a RBF the value of the function only depends on the distance to a fixed center c_i , usually Gaussian functions:

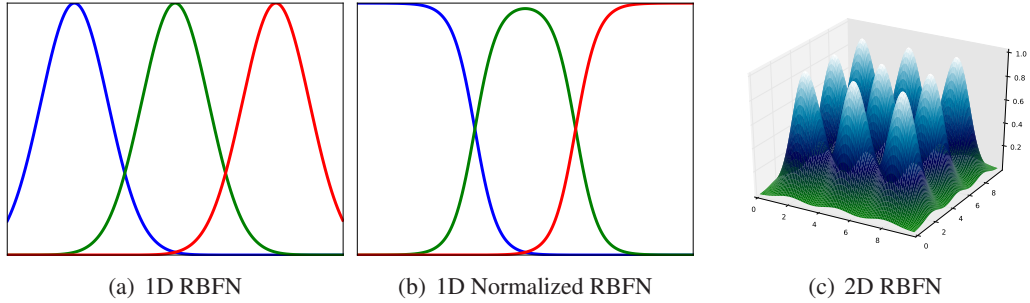


Figure 4.12: Example of Radial Basis Function Networks (RBFN).

$$\phi_i(x) = \phi_i(\|x - c_i\|) = \exp\left(-\frac{\|x - c_i\|^2}{2\sigma^2}\right) \quad (4.52)$$

Then, the centers c_i and the standard deviation σ are chosen so as to regularly cover the domain Ω of the target function. Moreover, it is usually convenient to normalize a RBFN. Then, for each $x \in \Omega$, the sum of the basis function ϕ evaluated at x is equal to one.

$$\tilde{\phi}_i(x) = \frac{\phi_i(x)}{\sum_{j=1}^M \phi_j(x)} \quad (4.53)$$

The basis functions are not radial anymore, but the whole basis better approximate Ω . Example of basis functions in 1D and 2D is given in Fig.4.12, where we can see the effect of normalization on a 1D case.

Some Results

Based on algorithm 4.6 and using Radial Basis Function Networks with Gaussian functions, we study the performance of approximation on the very simple 1D case. Each curve of Fig.4.13 is the average of the returns over 100 iterations. We used RBF Networks with isotropic kernels regularly spread out to cover the state-action space. In the first experiments, we use 50 basis functions, and 200 in the second one.

The results obtained are quite disappointing. The tabular version of Q-learning obtains better performance than the approximate ones. This is indeed not so surprising, since the size of the basis functions may not be big enough to capture the complexity of the target function Q^* to be approximated. Moreover, there are still two interesting observations one can make from these quantitative results. First, the approximate versions converge and the final performances of the approximation algorithm with 50 Gaussian kernels are lower than the one with 200 kernels which is itself lower than the tabular version; the higher the order of approximation, the better. Second, using 200 basis functions, the performances in the very first episodes are actually better than with the tabular version. Once more, this is not surprising. Reducing the dimensionality of the problem

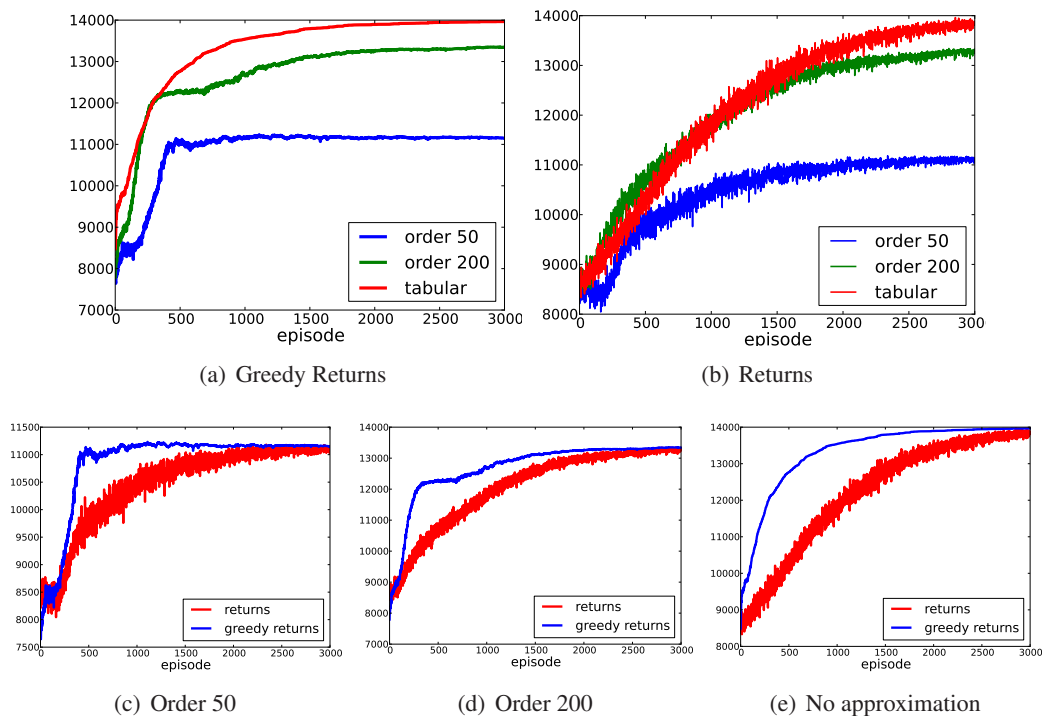


Figure 4.13: Learning Curve with Function Approximation.

increases the learning speed and the convergence is reached earlier. Thus, it seems possible to globally improve the convergence of the tabular Q-learning algorithm for shape grammar parsing by iteratively refining the degree of approximation as the episodes go. One could also combine this coarse-to-fine approach with a non-regular refinement of the approximation, allowing finer and more numerous kernels in the areas of the state-action space where the approximation needs to be accurate.

A completely approximate approach is somehow bound to fail though, since the objective of the approximation algorithm is to approximate the estimation of Q using a mean square criterion. Therefore, the approximation might be globally good over the whole state-action space, but not necessarily around its maxima. For that reason, the approximation poorly performs at providing the optimal parse.

Eventually, we believe that a more appropriate choice of the approximation basis as well as an adaptive coarse-to-fine approach could help using function approximation in the parsing problem. For instance, Tchebychev polynomials are often used for uniform (L_∞) approximation, and therefore might be a better choice of basis. However, we did not succeed in turning this very elegant concept of Reinforcement Learning into an efficient parsing tool.

4.4.3 User-Defined Constraints

So far we have presented how to automatically boost the performances of the 2D parsing algorithm. However, in practice, it is always convenient for the user to have some level of interaction with the program, both for the performances of the algorithm and for the feelings of the user. A way to interact with the parser would be to draw one or several elements of the facade. For example, a user could draw a rectangle specifying the position of the window. Let's now discuss how to take this additional information into account.

Fortunately, these hard constraints can be easily integrated in the RL framework. Let's suppose that the hard constraint is a labeled rectangle (x, y, w, h, c) with c its terminal symbol, x, y the position of the lower left corner, and w, h the dimensions. A label rectangle is an atomic shape $c(x, y, w, h)$. Since the agent aims at maximizing his return, that is to say his cumulative reward, we propose here to reward him more when he meets the constraint. For that matter, remember that the reward is computed based on the merit function and that is a signal emitted from the environment to the agent, consecutive to the agent's action. Therefore, when the agent takes a decision that creates the atomic shape $c(x, y, w, h)$, he receives a higher reward than he should have. For example, if the merit function is always between 0 and 1 the reward should be between 0 and wh . To enforce hard constraint, we set the reward to be $2wh$ when the agent takes a decision that exactly meets the constraint. In practice, if the agent puts an atomic shape, which intersection with the constraint is more than 95%, the constraint is considered as met and the reward is $2wh$.

This mechanism of hard constraints is indeed very easy to integrate in the RL framework. It does encourage the agent to meet the predefined constraint. However, these constraints are not as hard as expected, since it is still possible that the agent finds a more profitable parse which avoids the user defined constraint. In practice though they turn out to be very efficient and are particularly useful on real data (see chapter 5).

4.4.4 Minimum Description Length

One of the greatest assets of shape grammar is the compactness of its shape representation. However, sometimes the same grammar may express the same semantic segmentation of an image in different ways. As proposed by [Ripperda 2009], it is quite natural in this case to prefer the most compact solution. In this paper, they propose as criterion the minimum description length (MDL): the best representation of data is also the most compressed one.

The flexibility of the Reinforcement Learning framework also allows us to encode a MDL criterion very easily. We modify the objective of the learning agent: he aims at finding the sequence of rules that maximizes its returns while being as short as possible. This minimal length objective is standard in Reinforcement Learning. For instance, one can train a learning agent to find the shortest path to the exit in a maze by rewarding the agent with a fixed penalty (negative reward) as long as he has not found the exit of the maze. The more the agent remains inside the maze, the less he wins. Therefore he will tend to find the path that guarantees him to stay as little as possible in the maze: the shortest path.

1.	floorWall	\rightarrow_X	wall(x).floorWin + wall
2.	floorWin	\rightarrow_X	window(x).floorWall + wall
3.	floorWin	\rightarrow_X	tile*(x)
4.	floorWin	\rightarrow_X	(window(x).wall + wall)*(y)
5.	tile	\rightarrow_X	window(x).wall + wall

Table 4.2: A repeat grammar for parsing 1D floors. This grammar has both split and repeat rules. If the floor to be parsed is regular, then the representation is more compact with repeat rule than with binary split rules.

We adapt this classic example to our parsing problem. At time step t , the agent in s_t takes the decision a_t and receives a reward r_{t+1} . In addition to this immediate reward, he also receives a fixed penalty $-\rho$. For a given policy π , the agent builds a segmentation of the image $C \in L(G)$ by applying N rules and the complete return from the first state is now:

$$R_0(C) = \sum_{x \in \mathcal{I}} m(x, C(x)) - N\rho \quad (4.54)$$

Therefore, maximizing this cumulative reward fulfills the two objectives. Depending on the value of ρ , the optimal parse can be quite far from the expected one. If ρ is too big, then the agent prefers a very compact model regardless the quality of the semantic segmentation.

Extending BSGs

In order to underline the validity of the MDL penalty, we introduce here a Repeat rule (or operator) that somewhat changes the nature of binary split grammars. The repeat rule repeats a shape of same dimension an integer number of times. We denote repeat rules as:

$$\text{LHS} \rightarrow_{Ax} \mathbb{A}^*(x)$$

where \mathbb{A} is the atomic shape to be repeated along axis Ax with size x . Therefore, a repeat might create more than two shapes at a time: the parse tree is not necessarily binary any longer. However, we keep another important property: the repeat rule still carries a single parameter: the size of the repeated element.

Using these new rules, we propose a simple grammar to split 1D floors in table 4.2. This grammar can use binary split rules, repeat rules and then split rules, or a repeat rule in which a binary split rule has been nested.

Now let's suppose that we have a 1D image of a floor made of 3 regularly positioned windows. There exist several sequences of rules to express such a regular floor with the extended BSG, using from 2 to 6 rules.

1. 6 binary split rules: 1,2,1,2,1,2

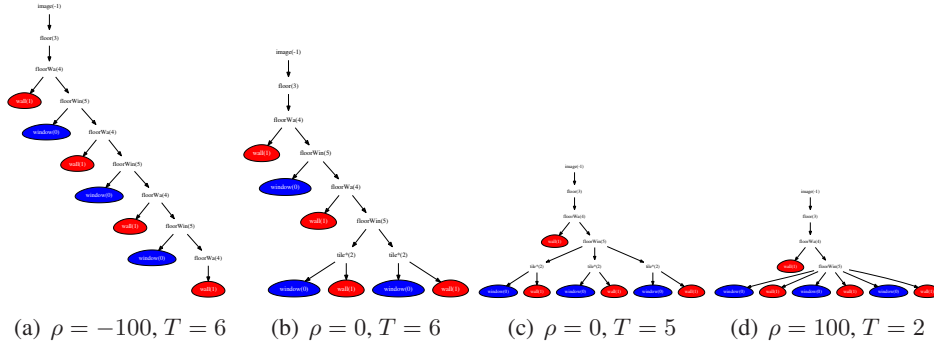


Figure 4.14: Different parse tree obtained for different MDL penalty ρ . We observe that without any penalty, the obtained solution can be any of the possible ones. With a negative penalty, the agent is encouraged to take as many decisions as possible whereas a positive penalty deters him from taking to many decisions. He tries to find the shortest path.

ρ	-10^4	-10^3	-10^2	-10	0	10	10^2	10^3	10^4
T	22	18	6	6	6	5 or 6	5	5	5
quality	55%	70%	100%	100%	100%	100%	100%	100%	70%

Table 4.3: Result of parsing with MDL penalty. ρ is the MDL penalty. The higher, the shorter is the decision process. T is the length of the decision process and *quality* is the percentage of well labeled pixels. In the experiments, the nest repeat rule 4 has not be used.

2. 5 binary split rules and 1 repeat rule: 1,2,1,3,5,5
3. 4 binary split rules and 1 repeat rule: 1,3,5,5,5
4. 1 binary split rule and 1 nested repeat rules: 1,4

However, we expect the most compact one to be the best one. We study here the effect of the MDL penalty ρ on the parsing result. The results are given in table 4.3, where we have removed rule 4 from the grammar to get more variability. Adding rule 4 drops the minimal number of rules to 2 for any positive penalty. In this table are presented for each value of the penalty ρ the horizon T (the number of decisions or the length of the decision process) as well as the segmentation quality of the output, which is equal to the number of well labeled pixels over the total number of pixels.

Note that if the penalty is too negative, then the agent is tore between taking as many decisions as possible (and therefore put as many windows as possible) and putting the wall and windows are the right locations. After a certain threshold, multiplying the number of elements becomes more

profitable than following the image, and therefore the agent tends to optimize only the topology of the tree rather than the image support. The same observation can be made for too large penalties. If we remove rule 4 from the grammar, then penalizing too hard the agent when he takes decisions pushes him to reach the end of the process as fast as possible, therefore creating as few windows and walls as possible on the input floor. However, the MDL penalty is not very sensitive, and one should really choose a large penalty (around 10^4) in absolute value before facing these degenerated cases.

Note that rule 4 of the extended BSG nests a binary split rule inside a repeat rule. Unlike the other types of rules presented so far, this nested rule has 2 parameters. In practice, the learning agent has to take two decisions at a time while applying such a rule. If each parameter can take 20 different values, then there are 400 possible of these nested rules. Such a number is still manageable, but does not fit the whole idea of the proposed algorithm, where a long sequence of simple decisions is preferred to a short sequence of very complex ones.

4.4.5 Grammar Selection

Shape grammars are very powerful representations. However, a common mistake while designing a shape grammar for automatic generation is to try to be as generic as possible. For instance, allowing the parameters of the binary splits to range from very small values to very high values may lead to unrealistic instances: high and narrow windows or very different sizes of windows on different floors for example. To solve this problem, one can use different grammars with different sets of parameters. A model is generated by first choosing one grammar and then deriving it. Since the rules of each grammar are consistent, the generated building is also realistic.

The same multiple grammars approach can be applied to parsing. A very simple way of doing so is to add a “meta-rule” 0 which chooses one of the possible grammars. All the following rules chosen by the agent are chosen among this grammar. Extending the derivation tree is once again quite natural and adapted to the proposed framework.

I do believe there would be other ways to do grammar selection (or model selection) more efficiently. For example, recent advances in Reinforcement Learning were made about multi-agent systems. We believe such an approach should give interesting results in grammar selection, and should be investigated in the future.

4.5 Conclusion

As a conclusion, this chapter introduced not only a very new and efficient parsing algorithm for 2D split grammars, but also a new framework: Reinforcement Learning. Although this whole theory is not very popular among the Computer Vision community, using it is not far-fetched at all: it constitutes the logical continuation of the previous work in shape grammar parsing in Computer Vision and of string grammar parsing. The first one was solved so far by Monte-Carlo while the latter one was solved by Dynamic Programming. Reinforcement Learning takes advantage of both worlds and produces a very elegant solution to Markov Decision Processes. Thus, formulating

the 2D parsing problem as a MDP naturally gives access to an entire world of algorithms, with well-known convergence properties. Through this formulation, we have proposed a new parsing algorithm that was proved to empirically converge on artificial data. This algorithm makes no assumption at all on the image to be parsed, and shows no restriction in term of topology.

So far, we have only explained how to parse artificial data, using a merit function that is either 0 or 1 for each pixel. In the next chapter, we are about to explain how to define merit functions on real data in order to parse facade image to achieve our goal: image-based procedural modeling of buildings.

Chapter 5

Top-Down Parsing: Applications

In chapter 4, we have presented a parsing algorithm that efficiently processes artificial images with BSGs. Working on artificial data is very convenient to understand the algorithm and the definition of the reward function is straightforward. In the current chapter, we explain how changing the reward functions enables us to parse real images. We propose different types of rewards in section 5.1 that corresponds to different types of grammars and different types of images. Section 5.2 demonstrates quantitatively the quality of the parsing algorithm on real data, while section 5.3 shows the qualitative results on different architectural styles and different grammars and shows the robustness of the proposed algorithm to occlusions and illumination conditions. Eventually, this last section provides single-view image-based procedural modeling of buildings, with several levels of modeling qualities, based on the power of the shape grammar formulation.

5.1 Defining Reward Functions on Real Images

So far, we have assumed that the RL parsing algorithm took as input a merit function m , which tells the learning agent about the quality of his segmentation decisions; the merit is directly related to the rewards. However, we did not explain how to get these rewards which in other optimization frameworks would have been called *data term*. In this section, we present different ways of building these functions, with different amount of user interaction, and on different kind of data. We will see that they take the form of posterior probability distributions.

5.1.1 Perfect Rewards for Artificial Data

In chapter 4, we have defined the merit function on artificial images. In that case, the input image presents perfect information, and since it was generated by a grammar, it can be parsed by the same grammar. Each pixel x of the image \mathcal{I} represents a symbol $\mathcal{I}(x) \in \mathcal{T}$. The merit function associated to this kind of data is a discrete Dirac distribution:

$$m(x, c) = p(c|x) = \delta(\mathcal{I}(x), c) \quad (5.1)$$

In the previous chapter, this merit function was already used with two colors (black and white) corresponding to two symbols (wall and window). Unfortunately having a perfect knowledge of the reward is not possible in real data. The basic idea behind estimating the appearance of each class is to perform some supervised learning on the terminal elements of the grammar. Depending on the kind of data, several approaches may be envisioned.

5.1.2 Randomized Forest for Consistent Architectural Styles

Some architectural styles are well characterized by the homogeneous appearance of their basic materials. When a style shows small appearance variations, having a supervised approach to learn the visual vocabulary of terminal symbols makes sense. We expect to find on new buildings what we have learned on the buildings of the training set. To do so, we must have both small variations and enough instances of buildings so as to ensure the generalization of the learning phase. Then a supervised learning approach builds classifiers that estimate the posterior probability of the symbol of a pixel $x \in \mathcal{I}$:

$$m(x, c) = p(c|x, \mathcal{I}) \quad (5.2)$$

Having pictures of a full street (rue Monge in the 5th district), we manually annotated 20 buildings. This amount of data is quite small but annotating images is very time consuming. Therefore these experiments were done as a proof of concepts.

In order to answer to the question of how to estimate the posterior probabilities based on the training data, we proposed a Randomized Forest approach, for the simplicity of its implementation and its classification performances. We describe here the principle of a Randomized Forest classifier.

Randomized Forests

Randomized forests [Breiman 2001] are very powerful classifiers. They have been adopted in a number of problems in Computer Vision like object recognition [Lepetit 2006], object classification coupled with bags of words techniques [Shotton 2008] or with graph cuts [Shotton 2007].

A randomized forest is used for supervised classification among C classes and is made of a set of T random decision trees. The leaves keep track of the visits of input feature vectors (see Fig.5.1). Internal nodes consist of a simple random test on a feature vector. When a feature vector associated to a label is dropped into the tree and reaches an internal node, it goes its way down to the left or the right child depending on the test response. After d tests, the vector falls into a leaf where the number of visits per label is updated. Thus, each leaf holds a histogram $h = (h_1, \dots, h_C)$ containing the number of feature vectors which have fallen in there for each of the C classes. One shall notice that in the process exposed above, some of the paths of the tree will not be explored

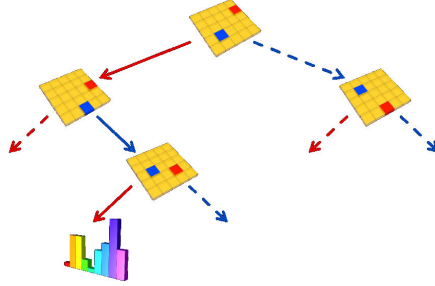


Figure 5.1: A Randomized Tree. The internal nodes are decision tests. If the red pixel is greater than the blue one, then it follows the red path to the left, and the right one otherwise. The leaf nodes store the visits for each class during training and are turned into posterior probability during testing after normalization.

by any of the vectors of the training set. Thus we dynamically create the nodes of the tree when needed. During the testing phase, an unlabeled feature vector is dropped in each tree of the forest. In a given tree τ , the feature vector falls in the leaf l_τ in which a histogram of labels is stored. Once normalized, this histogram provides an estimation of the posterior probability for the feature vector to belong to each class c , knowing the leaf l_t in which the patch has ended up:

$$p(c|l_\tau) = \frac{h_c}{\sum_i h_i} \quad (5.3)$$

Then, the probability over the forest is obtained by averaging the probabilities of all the trees.

$$p(c|x) = p(c|l_1, \dots, l_T) = \frac{1}{t} \sum_{\tau} p(c|l_\tau) \quad (5.4)$$

Typically, a randomized forest is made of 10 trees. In our case, we want to classify all the pixels of an image. The feature vectors we consider are patches centered on the pixels. Testing the classification results for different sizes of patches and different depths, we found that a depth of 15 and patches of sizes 13 are a good trade-off between memory usage and performances. Ultimately, we consider two kinds of decision tests: comparing the values of two pixels of the patch, or comparing the value of a pixel with a random threshold.

Examples of probability maps for each of the 7 terminal symbols of the Hausmann BSG as well as a pixel-wise segmentation based on the maximum a posteriori are presented in Fig.5.2.

Building a training set can be very time consuming in practice. Besides, the almost infinite diversity of real building appearances prevent us from scaling this method over a large database of buildings. As a result, we propose a second supervised method that does not require a complete data set of buildings, but only catches the appearance of the elements of the input images through very little user interaction.

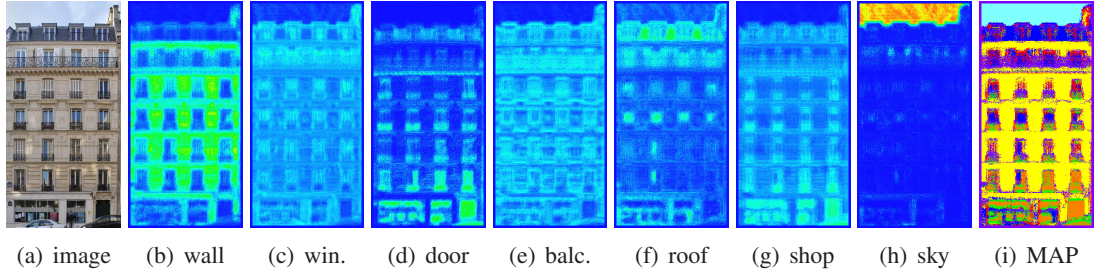


Figure 5.2: Merit functions for the Haussmann BSG. On the probability maps, the deep blue represents a probability 0 and red a probability 1. On the last image, the color of each pixel corresponds to the color of a symbol. Wall is yellow, window is red, door is orange, shop is green, sky is sky-blue, roof is blue, and balconies are purple. We can notice that some classes are well detected by the RF classifiers (walls, sky, roof) and some other are poorly detected (windows mainly). This is due to the complex appearance of windows that are often reflecting the scene in front of them.

5.1.3 Gaussian Mixture Models

The idea behind this method based on Gaussian Mixture Models (GMM) is that the user paints some brush strokes as examples of the appearance of terminal symbols. It could be a subset of terminal symbols or all of them. Thus, for each symbol $c \in \mathcal{T}$ we have a set of N_c observations (y_1, \dots, y_{N_c}) in \mathbb{R}^3 corresponding to the RGB values of the selected pixels. Following the ideas of [Blake 2004] in their grab cut segmentation method, we assume that for each class the observations are samples of an unknown mixture of Gaussian distributions made of K Gaussian kernels:

$$p(y|c) = \sum_{k=1}^K \pi_k \mathcal{N}(y|\mu_k, \Sigma_k) \quad (5.5)$$

where $\mathcal{N}(y|\mu, \Sigma)$ is the multivariate normal distribution of mean μ and covariance matrix Σ :

$$\mathcal{N}(y|\mu, \Sigma) = \frac{1}{(2\pi)^{3/2} |\det(\Sigma_k)|^{1/2}} \exp\left(-\frac{1}{2}(y - \mu_k)^T \Sigma^{-1} (y - \mu_k)\right) \quad (5.6)$$

The parameters of the Gaussian mixtures models for each symbol are estimated by the EM algorithm, first initialized by a K -Means. The EM algorithm alternates between evaluating soft assignments of data points to the Gaussian clusters, and then re-estimates the K Gaussian parameters μ_k, Σ_k and the mixture proportions π_k , until convergence. Then, the posterior probability is obtained by writing the Bayes' rule:

$$p(c|y) = \frac{p(y|c)p(c)}{\sum_{c' \in \mathcal{T}} p(y|c')p(c')} \quad (5.7)$$

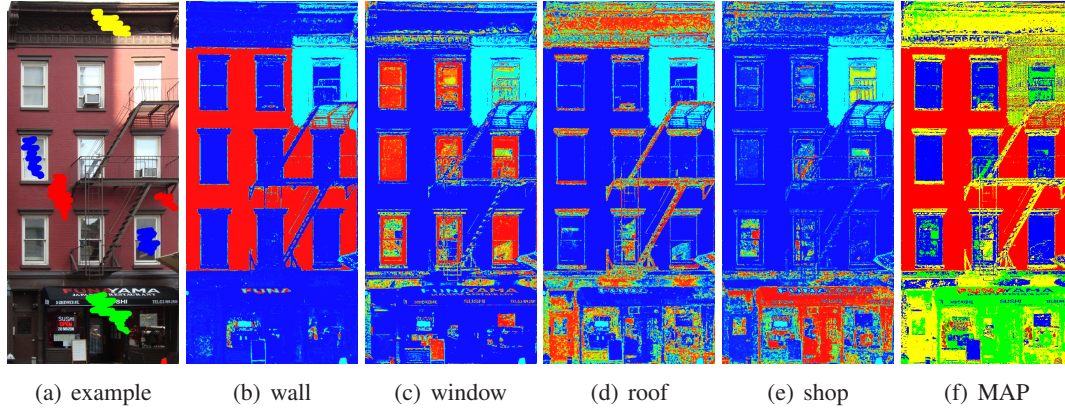


Figure 5.3: (a) shows the original image, in which some regions have been painted by the user, using the 4 colors: blue-window, red-wall, green-shop and yellow-roof. (b)-(e) are the learned merit functions using GMM for each of the terminal symbol of the 4-colors BSG. These probability maps range from blue (0.) to red (1). (f) the most probable symbol for each pixel, using the same color code as in (a).

Assuming that all the terminal symbols are equiprobable we obtain:

$$p(c_i|y) = \frac{\sum_{k=1}^K \pi_k^i \mathcal{N}(y|\mu_k^i, \Sigma_k^i)}{\sum_j \sum_{k=1}^K \pi_k^j \mathcal{N}(y|\mu_k^j, \Sigma_k^j)} \quad (5.8)$$

Another possibility would be to assume that the probability $p(c)$ is the frequency of symbols c in the training painted data: $p(c) \propto N_c$. For experiments we usually choose $K = 3$. Fig.5.3 shows the brushes strokes for the 4-colors BSG on an example as well as the posterior probabilities for each class, with a color code going from blue ($p = 0$) to red ($p = 1$). The GMM-based merit functions are usually providing a good data term since the appearance of an element amid a facade usually does not change much, unless it is occluded or received locally a specific illumination. As a conclusion, GMM-based rewards are a good trade-off between the user interaction level and the quality of the results, and have the great advantage of being usable on any image and any architectural style.

5.1.4 Unsupervised Hue based Rewards for Binary Parsing

So far we have proposed merit function based on supervised learning. The two previous methods need the intervention of a user, either to create a training set, or to paint some parts of the image. However, it could also be interesting to provide a merit function without supervision. Of course, this implies a small number of terminal symbols and some heuristic assumption behind.

For that matter, let's consider the case of binary segmentation of facade, such as described by the grammar given in table 2.2. It is equivalent to assert that we are parsing a facade into two classes: walls and windows. Then we make two assumptions:

1. There are more walls than windows on the facade.
2. The hue distinguishes the walls from the window.

Clearly, the first assumption is often violated in modern architecture, but is quite standard in more classic styles. The second assumption was proposed in [Liu 2010]. This assumption is very often satisfied. The reason is that the hue (H value in HSV color map) represents the pure color of a color, which is the color without shade. It first means that the hue is invariant to illumination. If the facade is half in the sun and half in the shadow, this illumination effect does not affect the hue image. Then, since the windows and the walls are usually showing different pure colors the hue should be different in these two regions.

Under these hypotheses, we first compute a Hue image and make a histogram of the hue values over the image. We expect to have two modes in the histogram, the heavier one corresponding to the walls and the other one to the windows. Under the assumption of Gaussian modes, the histogram is now made of two 1D Gaussian functions. Applying again the EM algorithm solves the problem. However, one should pay attention that the hue is an angle going from 0° to 359° . Therefore the computation of the means and the distances to the means should be taken with care, due to the circular nature of the histogram.

To compute a mean angle of $(\theta_1, \dots, \theta_n)$, the most efficient way is to compute it in \mathbb{C} :

$$\bar{\theta} = \arg(\bar{z}) = \arg\left(\frac{1}{n} \sum_{k=1}^n e^{i\theta_k}\right) \quad (5.9)$$

Besides, we also define the distance between two angles θ and ϕ as:

$$d(\theta, \phi) = \min(|\theta - \phi|, |\phi - \theta + 2\pi|) \quad (5.10)$$

After EM, we end-up with a single Gaussian model for the wall and a single Gaussian model for the windows:

$$p(x|wall) = \frac{\pi_{wall}}{\sqrt{2\pi}\sigma_{wall}} \exp\left(-\frac{(x - \mu_{wall})^2}{2\sigma_{wall}^2}\right) \quad (5.11)$$

$$p(x|window) = \frac{1 - \pi_{wall}}{\sqrt{2\pi}\sigma_{window}} \exp\left(-\frac{(x - \mu_{window})^2}{2\sigma_{window}^2}\right) \quad (5.12)$$

These likelihoods can be now turned into posterior probabilities exactly by the same process as in the GMM merit function.

An example of image of facade, its hue image, the most probable symbols at each pixel and the histogram of hue both in linear and polar scales are shown in Fig.5.4. We clearly see that the two terminal symbols have very different hues, and therefore this heuristic criterion is very often satisfied. Results are presented and commented in section 5.3.

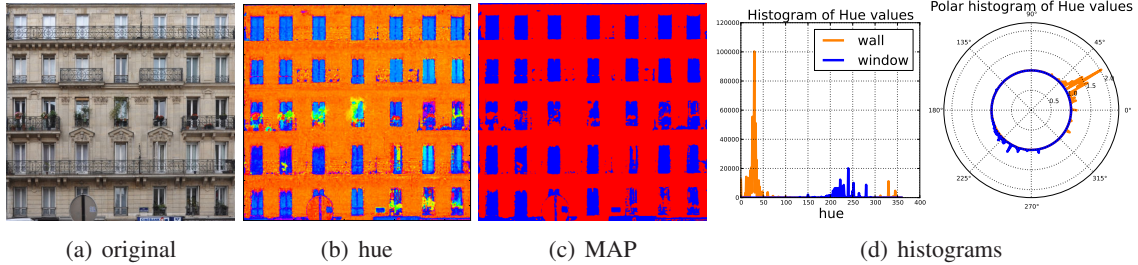


Figure 5.4: Analyzing a facade through the Hue. Two modes clearly appear: one for the wall and the other for the windows.

5.1.5 Depth Maps

The rewards or merits proposed so far are all dealing with optical data. However, there is absolutely no reason why we should restrict ourselves to such data, all the more since in Computer Vision, 3D data are often available through acquisition or retrieved by structure-from-motion [Snavely 2006, Snavely 2008]. We will here present a way to create a depth map of a facade that can serve as input of a 2D parsing, with depth information.

We assume now that we have N images of the facades. Using the structure-from-motion tool Bundler [Snavely 2006, Snavely 2008], we obtain a 3D points cloud of the facade. Considering that this points cloud represents a facade, the data should coarsely correspond to a plane, with some 3D points in front of it (balconies and shops) and some 3D points lying behind the facade plane (roof and windows).

In order to find this plane, we apply a RANSAC method [Fischler 1981]. Sampling 3 3D points (x_i, y_i, z_i) , we can compute a plan $\Pi : ax + by + cz + d = 0$, with $n = (a, b, c)^T$ the normal vector of Π of length 1. Assuming that $d \neq 0$, we can solve $\alpha x + \beta y + \gamma z + 1 = 0$ by inverting the linear system:

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = - \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad (5.13)$$

Then, normalizing the α , β and γ gives the plane equation that goes through the three points:

$$\begin{aligned} a &= \frac{\alpha}{\sqrt{\alpha^2 + \beta^2 + \gamma^2}} & b &= \frac{\beta}{\sqrt{\alpha^2 + \beta^2 + \gamma^2}} \\ c &= \frac{\gamma}{\sqrt{\alpha^2 + \beta^2 + \gamma^2}} & d &= \frac{1}{\sqrt{\alpha^2 + \beta^2 + \gamma^2}} \end{aligned} \quad (5.14)$$

Then, the distance between a 3D point $X = (x, y, z)^T$ and the plane Π is given by:

$$d(X, \Pi) = |ax + by + cz + d| \quad (5.15)$$

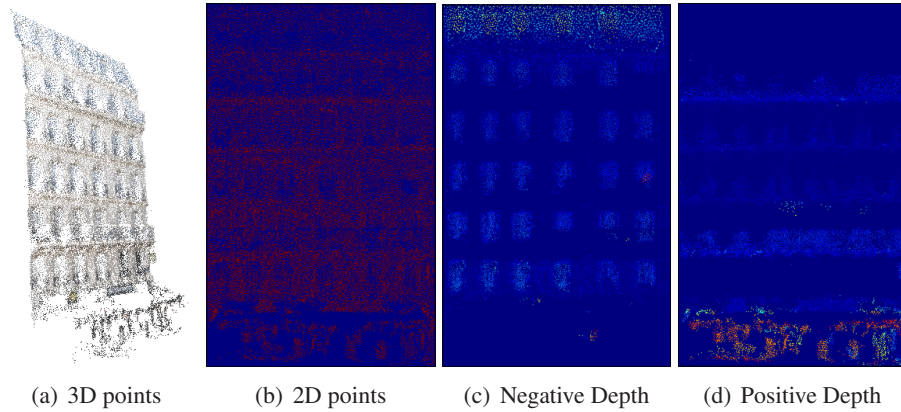


Figure 5.5: Creating a Depth Map from a 3D Points Cloud.

Using a threshold ϵ , one can easily compute how many points of the points cloud agree with the plane Π . After a couple of RANSAC iterations, we obtain a set of inliers (X_1, \dots, X_k) corresponding to points belonging to the facade plane, and we find the plane that best explains these k inliers using a Principal Component Analysis (PCA), through a Singular Value Decomposition (SVD). We write $X_i = (x_i, y_i, z_i, 1)^T$, and A the matrix obtained by stacking all the X_i . The SVD of $M = A^T A$ gives a new coordinates system P . The two eigenvectors associated to the two highest eigenvalues lie on Π and define a orthogonal coordinates system of the plane, whereas the last coordinate defines the distance to the plane.

The next step is to find a bounding box that is aligned with the facade vertical and horizontal directions and rotate the plane coordinate system so that it is aligned with this bounding box. Ultimately, we discretize this plan and obtain an image D for which each pixel x corresponds to a depth $D(x) = d$ that can be positive or negative. The image might not be completely filled, since there might be some points of the facade that do not correspond to the orthogonal projection of a 3D point of the cloud. Moreover, if two 3D points fall into the same pixel, the value of this pixel is the one with lower absolute value, so as to get rid of outliers lying very far from the facade plane. We note E the Boolean image that is $E(x) = 1$ if there exists a 3D point that projects itself onto x and 0 otherwise. Such an image is given in Fig.5.5, along with the depth maps (split into positive and negative).

In order to get a merit function, we split the set of terminal symbols $\mathcal{T} = \mathcal{T}^+ \cup \mathcal{T}^- \cup \mathcal{T}^0$. The symbols in \mathcal{T}^+ should correspond to extrusion (for examples shops and balconies), the symbols in \mathcal{T}^- correspond to intruded symbols (windows, roof) and the last set \mathcal{T}^0 represents the symbols that lie on the facade plane (walls and door). The merit function is then easily computed as using the depth map:

$$\begin{aligned}
m(x, c) &= 0 && \text{if } E(x) = 0 \\
&= 1 && \text{if } c, D(x) \in \mathcal{T}^+ \times [\epsilon, \infty[\\
&= 1 && \text{if } c, D(x) \in \mathcal{T}^- \times [-\infty, -\epsilon[\\
&= 1 && \text{if } c, D(x) \in \mathcal{T}^0 \times [-\epsilon, \epsilon] \\
&= 0 && \text{otherwise}
\end{aligned} \tag{5.16}$$

Let's now evaluate the performances of the Reinforcement Learning based shape grammar parsing algorithm on real images, using some of the previously defined reward functions.

5.2 Quantitative Validation

Let's first study the empirical convergence of the parsing algorithm on real data, before measuring the parsing quality. Since this algorithm optimizes both the topology and the geometry of the shape with respect to the image cues, we propose two different similarity measures, to compare the ground truth parse with the optimized one. The first one is a measure of segmentation and the second one a measure of topology.

5.2.1 Convergence on Real Data

We have already presented learning curves of Q-learning agents with different exploration policies in Fig.4.9 and Fig.4.11 on artificial data and on the binary segmentation BSG. Here, we run the same kinds of experiments on real data and different grammars. We associate each BSG with a merit function to be applied on real images: the binary segmentation BSG is associated with the Hue reward, the 4-colors grammar with GMM reward and the Hausmannian grammar with Randomized Forest reward. For each couple grammar-reward, we take an image that can be parsed with the grammar, and parse it 100 times with a Q-learning agent, learning for 3000 episodes, with a data-driven exploration. Then we obtain the learning curves by averaging at each iteration the 100 returns and greedy returns. The learning curves are shown in Fig.5.6.

As in the artificial case, the proposed algorithm does converge. The average learning curves are increasing and smooth. The experiments seem to confirm the theory. Furthermore, we observe that the more complex the grammar, the more difficult the learning is. For example, in Fig.5.6(a), the average curves are thin, which means that all the experiments were quite similar, whereas in Fig.5.6(c), the curves are much more noisy, which means that the experiments showed more variability. Besides, the greedy curve of this grammar is not always increasing. This phenomenon is quite standard in Reinforcement Learning and the explanation is the following: the initial estimates of the Q function are 0 for all state-action pairs. At the beginning, the agent chooses randomly some actions, and therefore updates the estimate of Q for state-actions pairs that were visited. Since the rewards are positive, the greedy actions at the beginning, always correspond to the actions that were visited which is in average misleading for the agent. He tends to follow these decisions,

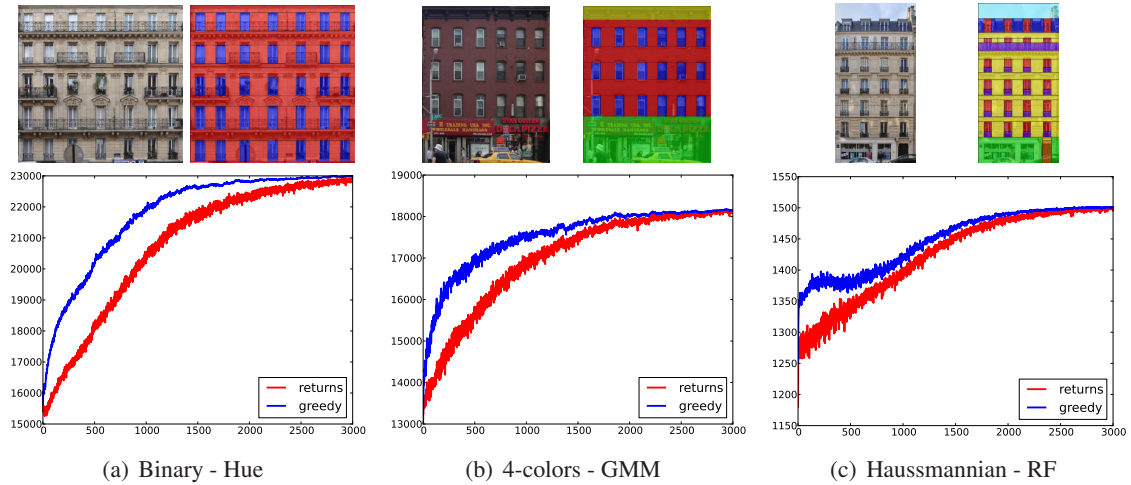


Figure 5.6: Convergence of the algorithm on real data. The curves are averaged among 100 experiments.

while they are completely random, and it leads to poor returns. After some time, the agent has a better knowledge of the decisions through exploration, and he can therefore follow truly better paths. That is why, the curve first increases, because random actions are better than nothing, then decreases because the agents trusts bad decision at first, and finally increases, because the level of exploration is sufficient so that he does not blindly follows bad decisions.

In our case, the learning curves for the binary grammar and the 4-colors grammar do not show this initial decrease. There are two reasons for that. First, the grammar is simple enough so that this appears in the very first iterations, and the amplitude of the phenomenon is similar to the amplitude of the noise of the curves. Second, the agent follows a data-driven exploration policy which makes him follow right away good decisions with a higher probability, thus decreasing the amplitude of this phenomenon. On the contrary, the Haussmannian grammar and the images are much more complex. It means that the agent learns more slowly and that the data-driven proposals may be wrong. In that case, the phenomenon appears with high amplitude, and not immediately in the very first iterations.

The convergence speed depends both on the complexity of the grammar and the topology of the building to be parsed. Indeed, in a complex grammar, the agent has to choose between many options at each decision: the more options the harder to learn. Furthermore, the topology of the buildings is directly linked to the horizon T of the decision process, since each element of the facade corresponds to a decision of the agent. As a consequence, it is not surprising to observe slower learning speed on complex grammars and on huge facades. A good way to speed-up the convergence while parsing huge facades (like skyscrapers) is therefore to decrease the complexity of the grammar. For instance, the use of repeat rules (see section 4.4.4) may decrease the number

of decisions by constraining the layout.

Now that we have presented and explained the empirical convergence of the parsing algorithm, let's focus on the segmentation performances and how to measure them.

5.2.2 A Measure for Segmentation

As mentioned in the previous sections, a parse corresponds to a semantic segmentation of the image. Each pixel receives as label the symbol of the terminal shape it belongs to. Let's denote $\mathcal{T} = \{c_1, \dots, c_n\}$ the finite set of n terminal symbols of the shape grammar G . There are basically two ways to compare the ground truth segmentation with the one found by the algorithm: the first one is the detection rate and the second one is the confusion matrix M of size $n \times n$.

The detection rate is a very global measure that represents the percentage of correctly labeled pixels by the algorithm compared to the ground truth. This measure is too generic, since it does not give any information on the different classes.

The confusion matrix M is a more detailed measure. The element M_{ij} on row i and column j of M represents the percentage of the pixels that belong to class c_i in the ground truth, and that were labeled c_j by the parsing algorithm. A perfect confusion matrix would be the identity matrix, since it would mean that all the pixels have been correctly labeled. Although the confusion matrix describes the quality of the segmentation better, it still has two main drawbacks. The first one is that it completely gets rid of the structure of the segmentation, since it is completely pixel-wise. The second is that it does not take into account the size of each class or symbol.

To understand the second drawback, imagine that a class c_1 is supposed to represent a very small structure of the building, for example one rectangle of size 10×10 , and that class c_2 represents a bigger structure, a rectangle of size 100×100 . Let's suppose that the algorithm misses by two pixels in each direction the rectangles. In class c_1 , 64 pixels will be correctly labeled and in class c_2 $98 \times 98 = 9604$ pixels will be correctly labeled. Therefore, on the diagonal of the confusion matrix, on the line c_1 we will have 64% and 96.04% on line c_2 . The geometrical error is the same, but the two measures are very different, leading to an unfair comparison.

However, a geometrical measure is neither obvious to build nor to interpret because the topology of the ground truth and the optimal parse may be different as well. For that matter, we propose a topological measure between the ground truth parse and the optimal one.

5.2.3 A Measure for Topology

This measure aims at capturing the topological and semantic similarity between two parse trees. The idea is to compare them recursively.

Let c be the symbol of the root of a tree T and t^1, \dots, t^n its children (that are themselves trees). Besides, we will note $\#T$ the number of nodes of the tree T . Now let's suppose that we have

two trees T_1 and T_2 . All the previous notations are indexed now by 1 or 2. Then, we define the similarity η between the two trees as:

$$\eta(T_1, T_2) = \frac{\delta(c_1, c_2)}{\#T_1 + \#T_2 - 1} + \sum_{i=1}^{\max(n_1, n_2)} \eta(t_1^i, t_2^i) \frac{\#t_1^i + \#t_2^i}{\#T_1 + \#T_2 - 1} \quad (5.17)$$

where, δ is the Kronecker operator. Thus, the similarity is bounded between 0 and 1. Note that there would be other ways to build a similarity between two trees based on the same kind of recursive principles. However, this definition is convenient, since a similarity η of 0 means that the nodes of the two trees are different everywhere, whereas a similarity of 1 means that the two trees exactly carry the same symbols at the same positions in the trees. Weighting the similarities by the number of nodes of each subtree allows us not to penalize too hard bad branches close to the root if they do not carry much information. This measure does not depend on the depth of the nodes, but on the number of nodes carried by the subtrees, which is much more robust.

Now that we have a way to measure the quality of a parse tree both in terms of segmentation and topology, let's see how the parsing algorithm performs on several kinds of data and with several grammars.

5.2.4 Paris Benchmark 2010

We proposed this benchmark for a semantic segmentation method using procedural shape prior [Teboul 2010]. For this benchmark, we have 20 buildings annotated for training, and 10 annotated for testing. Note that the annotations were not done using a grammar. There are two main consequences: first, using a topology measure is impossible, since the ground truths are not expressed as parse trees. Then, it is impossible to get a detection rate of 1, or equivalently a diagonal confusion matrix in this case. Despite, these two major drawbacks, this benchmark was at some point the only existing one, which justifies using it.

A Randomized forest classifier (see section 5.1.2) made of 10 trees of depth 12, and working on feature vectors of dimension 39 (RGB patches of size 13) is trained on the 20 images, and used directly as a merit function. We ran the RL parsing algorithm on the 10 images for 3000 episodes using the complex Hausmannian grammar (see section 2.4) made of 281 rules. In average, the agent has to take around 80 decisions per episode.

We compare the confusion matrix obtained on the 10 images using the proposed Q-learning based parsing algorithm, with 3 confusion matrices coming from 3 other semantic segmentation methods: MAP, a Potts model and the method from [Teboul 2010].

MAP classification is obtained by building a segmentation of the images based on the Randomized forest classifier only. To do so, each pixel x is assigned to the class c the maximizes the posterior probability $p(c|x)$ provided by the Randomized Forest:

$$\forall x \in \mathcal{I}, \quad c^* = \arg \max_c p(c|x) \quad (5.18)$$

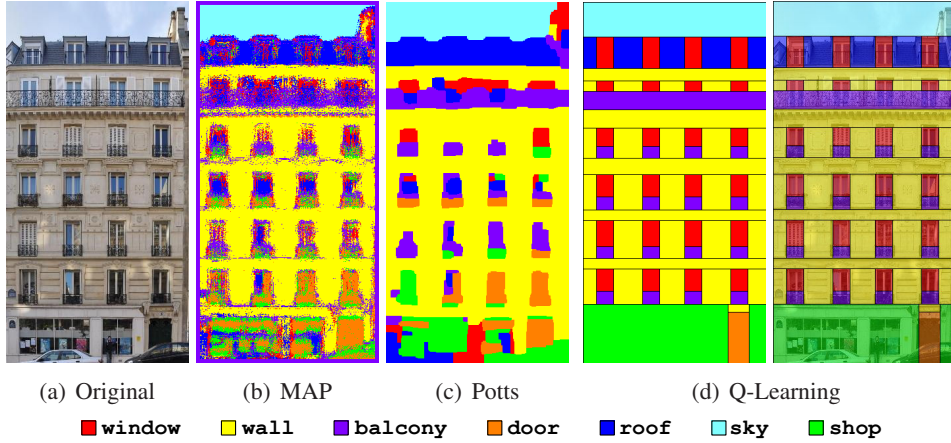


Figure 5.7: Comparing the segmentations obtained by 3 different methods.

Obviously, this segmentation method is very sensitive to noise and produces very inconsistent segmentations (see Fig.5.7(b).) Thus, it appears fairer to compare the proposed method with a smoother segmentation using a Potts model.

Potts Model is a very well-known smooth graphical model. It defines the following energy on the image graph:

$$E(C) = \sum_{x_i \in \mathcal{I}} -\log p(C(x_i)|x_i) + \lambda \sum_{x_i \in \mathcal{I}} \sum_{x_j \sim x_i} \mathbf{1}(C(x_i) \neq C(x_j)) \quad (5.19)$$

where C is a semantic segmentation; it maps every pixel x to a class $C(x) \in \mathcal{T}$. Thus, neighboring nodes with different labeling will be penalized with a constant λ whereas no penalization is given when the same labels are given to these nodes. In Fig.5.7, we can see that the segmentation obtained by a Potts model is smoother than the MAP segmentation. However, it neither guarantees any structure of the final segmentation nor any increase of the detection rates.

$\begin{pmatrix} \mathbf{29} & 13 & 13 & 11 & 22 & 6 & 6 \\ 4 & \mathbf{63} & 11 & 8 & 4 & 2 & 8 \\ 10 & 11 & \mathbf{42} & 13 & 12 & 1 & 11 \\ 2 & 2 & 1 & \mathbf{90} & 0 & 0 & 5 \\ 8 & 12 & 5 & 0 & \mathbf{62} & 12 & 0 \\ 1 & 0 & 0 & 0 & 4 & \mathbf{95} & 0 \\ 6 & 7 & 9 & 43 & 8 & 1 & \mathbf{26} \end{pmatrix}$	$\begin{pmatrix} \mathbf{29} & 16 & 12 & 11 & 23 & 3 & 6 \\ 2 & \mathbf{72} & 8 & 7 & 2 & 1 & 8 \\ 6 & 11 & \mathbf{60} & 10 & 4 & 0 & 9 \\ 0 & 1 & 1 & \mathbf{96} & 0 & 0 & 2 \\ 5 & 12 & 1 & 0 & \mathbf{71} & 11 & 0 \\ 1 & 0 & 0 & 0 & 3 & \mathbf{96} & 0 \\ 6 & 5 & 6 & 46 & 7 & 1 & \mathbf{29} \end{pmatrix}$	<i>window</i> <i>wall</i> <i>balcony</i> <i>door</i> <i>roof</i> <i>sky</i> <i>shop</i>
MAP	Potts, $\lambda = 1$	

Random Exploration with Procedural Shape Prior was proposed in [Teboul 2010]. This method has indeed several drawbacks. First it is slow; about 10^6 buildings have to be generated before convergence. Then, there is no guarantee of convergence, and in practice this algorithm gets often stuck in local minima. However, this method does provide a segmentation that belongs to the language of the input shape grammar, which guarantees a structure of the output building.

$\begin{pmatrix} \mathbf{81} & 9 & 6 & 0 & 4 & 0 & 0 \\ 5 & \mathbf{83} & 8 & 1 & 0 & 0 & 3 \\ 13 & 13 & \mathbf{72} & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & \mathbf{71} & 0 & 0 & 29 \\ 8 & 12 & 0 & 0 & \mathbf{80} & 0 & 0 \\ 2 & 0 & 0 & 0 & 4 & \mathbf{94} & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & \mathbf{95} \end{pmatrix}$	$\begin{pmatrix} \mathbf{81} & 11 & 3 & 0 & 5 & 0 & 0 \\ 5 & \mathbf{84} & 7 & 1 & 1 & 0 & 2 \\ 10 & 26 & \mathbf{63} & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & \mathbf{84} & 0 & 0 & 14 \\ 10 & 4 & 0 & 0 & \mathbf{86} & 0 & 0 \\ 2 & 0 & 0 & 0 & 4 & \mathbf{94} & 0 \\ 0 & 1 & 0 & 2 & 0 & 0 & \mathbf{97} \end{pmatrix}$	<i>window</i>
		<i>wall</i>
		<i>balcony</i>
		<i>door</i>
		<i>roof</i>
		<i>sky</i>
		<i>shop</i>
[Teboul 2010]	RL Parsing	

As expected, the confusion matrix of the MAP segmentation is quite noisy, with poor detection rates on the diagonal. Moreover, the Potts matrix is not much better than the MAP one. Although the segmentation itself is smoother, the confusion matrix is very noisy. This is due to the absence of structure of the Potts model. Unsurprisingly, the two last confusion matrices are much smoother, and show better detection rates.

The reason for these better results is the use of the procedural shape prior that introduces some context and resolve the potential ambiguities between classes. For instance, windows often look like the sky, because they are reflecting it. A MAP segmentation or a Potts one would tend to label as sky the pixels in the window. However, a shape grammar completely forbids such a configuration; the sky cannot be lying in the middle of the facade. Thus, the confusion matrices of shape grammar based methods are bound to be better.

To facilitate the comparison between these two confusion matrices, the diagonal elements of the confusion matrix of the proposed method are green if the detection rate is greater or equal, and red if it is lower. The first thing to notice is that the two confusion matrices are quite similar. This is due to the fact that the two methods optimize the same functional. The slight increase in the detection rate of the RL method is mainly due to better convergence properties of this method. Indeed, the method we proposed in [Teboul 2010] needs around 10^6 generations and about 10 minutes to converge, whereas the RL method finds a solution after only 2000 iterations and around 30 seconds, and avoids local minima better. Eventually, the slight decrease of detection in the balcony is due on the one hand to a more complex grammar (a running balcony can appear at any floor, whereas in [Teboul 2010], the grammar enforces a running balcony to appear on the second and the last floor), and on the other hand, the balconies are often small structures, and as mentioned earlier, the detection rates for small structures are more sensitive to small variations than big structures.

This benchmark is quite limited, but already shows the benefits of using shape grammar to segment complex and structured geometries such as facades, and also the performances of Reinforcement Learning to guide the search in high dimensional spaces. In the next section, we propose

	mean	std
topology	0.93	0.09
appearance	0.81	0.07

Table 5.1: Statistics of the performances on the full Monge dataset, using the same Randomized Forest classifiers.

a more complete database of Haussmannian facades in order to evaluate the performances of the proposed algorithm on a larger scale, with respect to the topology and the appearance of the hierarchical segmentations.

5.2.5 Paris Benchmark 2011

In the previous benchmark, over the 104 images taken from rue Monge in the fifth district of Paris, we kept only 20 for training and 10 for testing. However, testing on 10 images is very limited and is definitely not enough to assess the performances of the parsing algorithm. For that matter, we have manually annotated the full data set, which consists of 104 facade images, with the Haussmannian shape grammar. Since the Randomized Forest classifiers are not very informative and do not generalize very well on the whole training set, we process the data in the following way: for each image, we run 5 Q-learning agents, parsing the image for 5000 episodes with a data-driven exploration and the Randomized Forest based rewards. We compute the topological similarity and the detection rates of the 5 semantic segmentations found, and keep the one that maximizes the global detection rate (percentage of well labeled pixels). Then we show the confusion matrix on the 104 buildings, as well as the statistics of the topological similarities.

$\left(\begin{array}{cccccc} \mathbf{27} & 15 & 15 & 13 & 19 & 4 & 8 \\ 5 & \mathbf{63} & 11 & 9 & 4 & 2 & 7 \\ 11 & 17 & \mathbf{34} & 13 & 12 & 2 & 11 \\ 2 & 2 & 1 & \mathbf{81} & 3 & 0 & 10 \\ 10 & 6 & 11 & 4 & \mathbf{54} & 10 & 4 \\ 4 & 3 & 3 & 1 & 14 & \mathbf{75} & 1 \\ 6 & 12 & 10 & 42 & 7 & 0 & \mathbf{22} \end{array} \right)$	$\left(\begin{array}{cccccc} \mathbf{68} & 23 & 4 & 0 & 4 & 2 & 0 \\ 3 & \mathbf{87} & 7 & 0 & 1 & 0 & 1 \\ 9 & 24 & \mathbf{64} & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & \mathbf{53} & 0 & 0 & 46 \\ 6 & 3 & 0 & 0 & \mathbf{83} & 8 & 0 \\ 1 & 0 & 0 & 0 & 3 & \mathbf{96} & 0 \\ 0 & 6 & 1 & 6 & 0 & 0 & \mathbf{88} \end{array} \right)$	<i>window</i>
		<i>wall</i>
		<i>balcony</i>
		<i>door</i>
		<i>roof</i>
		<i>sky</i>
		<i>shop</i>
MAP	RL Parsing	

First, we can observe a significant decrease of the detection rates of the windows, the balconies and the doors, compared to the small dataset of 10 buildings. This is not very surprising, since they are the classes with the highest variability and also they represent the smallest elements of the facades. Therefore, it is quite often that the parsing algorithm actually finds the windows but misses some pixels of it, leading to important drop in the detection rates.

The second observation is that the parsing algorithm outperforms the MAP segmentation based on the Randomized Forest only, except for the specific case on doors. This can be explained by paying attention to the last line of the first confusion matrix. Shops are more labeled doors than shops in a MAP classification. Doors are detected everywhere on the ground floor. It means that in the parsing algorithm, the agent will be tempted to put a door anywhere on the ground floor, and not specifically on the real door.

Furthermore, we also observe that the Randomized Forest trained on 20 buildings does not generalize well on the full dataset. For some buildings, almost no windows are detected, and therefore it makes it harder for the parsing algorithm that can only rely on a few classes such as wall, roof or sky. No question that a better data term, such as a different GMM classifier for each facade would boost the performances of the parsing that could rely on sensible information.

Eventually, we would like to stress that even when the semantic segmentation is not perfect, the topology is almost always perfect, with 0.93% of similarity. Note that these 7% mostly come from a phenomenon that occur at the end of the floors or at the end of the facade, when the parser cut the last wall into two. It actually gives the same reward to put a single wall or 2 walls, but the agent usually puts two, whereas in the ground truth, the last wall is one and only one wall. Since this small topological difference appears at all the floors, it can impact on the global topological similarity between the ground truth and the optimal parse, which explains why the mean is not closer to 1.

The conclusion of these massive tests on a bigger database is that the parsing algorithm performs surprisingly well, even when the data term (reward) is poorly informative and very noisy as it is the case here. The low data term quality is compensated by the power of shape grammars and a method to parse them properly. However, there are some cases for which the data term is so bad that the agent is completely lost, and the optimal parse he finds brings him a better reward than what he would get by following the ground truth. This raises the question of learning a proper data term based on example, so that the optimal parse coincides with the expected parsing of the image. Such a question is left to future work. For the time being, a good alternative to solve this problem is indeed to switch the rewards, from a Randomized Forest one to a GMM one, in order to get a better data term on a specific image. Unfortunately, this approach is not appropriate for big databases like this one, since the user has to manually paint some brush strokes in every single image of the database, which is a slow and tedious process.

5.2.6 Robustness to Noise

Another important aspect is the robustness of the segmentation to noise. In order to assess the performances of the proposed algorithm, we take an image of a facade that can be expressed with the 4-colors grammar, and learn a GMM classifier on the non-corrupted original image. Then, we increasingly add salt-and-pepper noise, corrupting from 0 to 100% of the pixels by painting them either in black or in white. Since the GMM has been trained on the original image, the corrupted pixels are necessarily misclassified by the GMM, and they receive a uniform posterior probability to belong to any of the 4 classes. Therefore the relevant information brought by the data term is

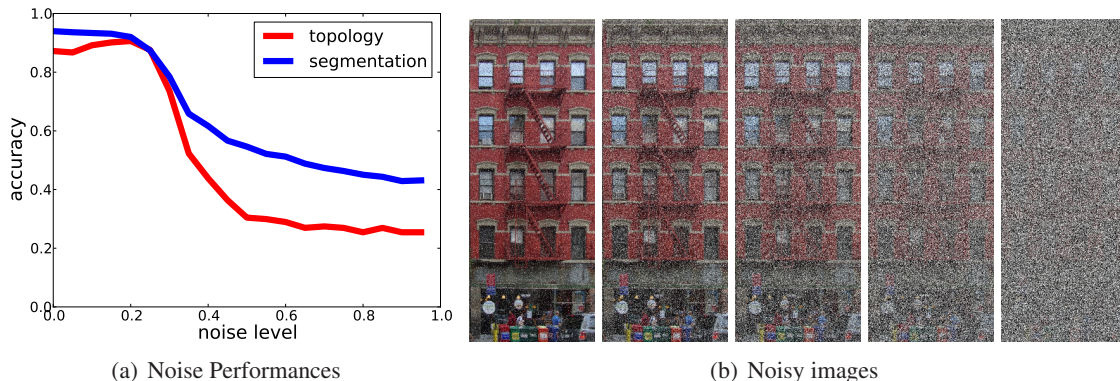


Figure 5.8: Evolution of the detection rates with increasing level of salt-and-pepper noise. The experiments were done with 20 noise corruptions ranging from from 0% to 95% of the pixels. In (b), we only display a noise corruptions 10%, 30%, 50%, 70% and 90% as illustration.

corrupted in the same proportions than the image is corrupted by the salt-and-pepper noise.

Then, for each level of noise, we run 20 times the RL algorithm for 2000 episodes, with a ϵ -greedy exploration policy, and store the segmentation performances. We show in Fig.5.8 the evolution of the average detection rate over the 20 parses with the level of noise.

As show in Fig.5.8, the proposed method is quite robust to noise. The red curve represents the average topological similarity measure between the ground truth and the optimal parses. The blue curve refers to the average segmentation similarity (global detection rate).

The first comment, is that the performances start to drop after 30% of pixels corrupted by a salt-and-pepper noise. This level of noise is underestimated, since the experiments where run on real data with a GMM classifier which is itself not perfect and quite noisy. As a consequence, even with no additional salt-and-pepper noise, the data term (merit function) is noisy by nature.

The second observation is that for very high noise levels, the image support is somehow lost: the merits are random. However, we do not obtain 0% of detection rate or 0% of topology similarity, because the parsing still produces a random building, with no link with the image, but if still belongs to the language of the grammar. Thus, this random optimal building shares to a certain extent a common topology with the ground truth. From the segmentation point of view, we obtain detection rate greater than random even if the building is built randomly. A completely random segmentation of the image would independently put for each pixel the good label with probability 0.25 (we are using the 4-colors grammar). Then the global detection rate would also be 0.25. In our case, the building is built randomly, but the labeling of the pixels are not independent. Therefore the building we obtain is more likely to be similar to the expected one, and the probability is here around 0.4;

So far we have measured the quality of the algorithm on different kinds of data and different kinds of grammars according to several criteria: convergence, segmentation quality, respect of the topology and robustness to noise corruption. In the next section, we discuss the qualitative

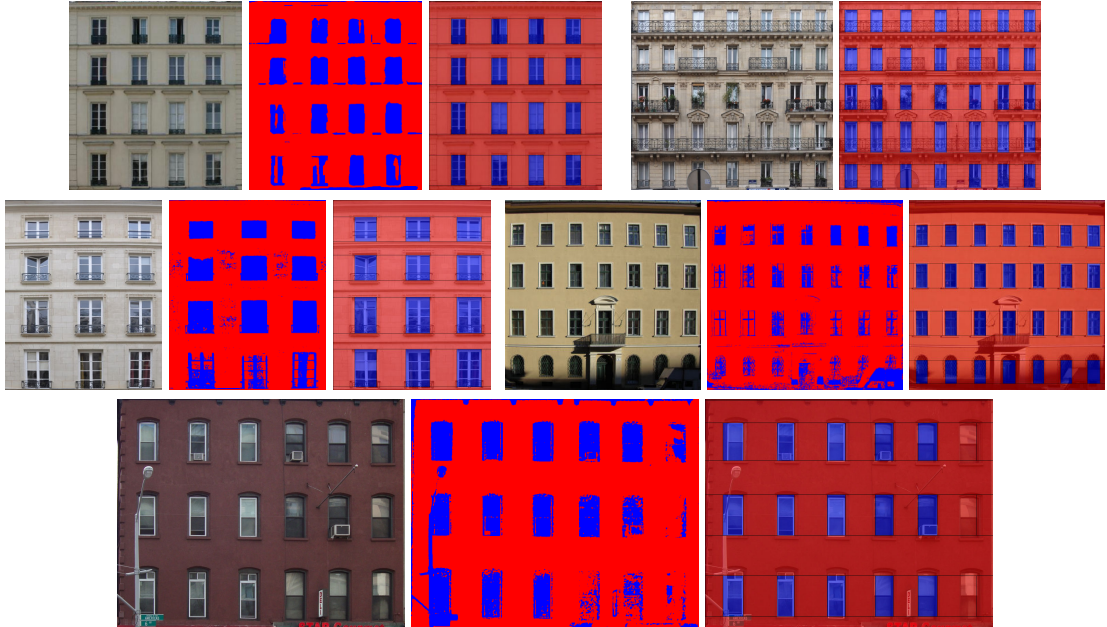


Figure 5.9: Parsing images with binary segmentation grammar based on Hue rewards.

performances of the parsing algorithm, and confront it to several grammars, several architectures and several challenging conditions.

5.3 Qualitative Validation

So far, we have focused on measuring the performances of the proposed algorithm, which outperforms the other existing methods. Now, we discuss the qualitative results on different types of architectures with different kind of BSGs.

5.3.1 Binary Segmentation

The binary segmentation grammar presented in section 2.4 is the simplest grammar. It separates the walls from the windows. We now give some examples of parsing this grammar on different types of images, using a hue-based reward. Therefore such a parsing is completely unsupervised.

The Hue reward seems to provide quite good merit functions on some images, but turns out to be poorly separating the walls from the windows on others. For example, on the last image of Fig.5.9, the last column of windows is almost not detected. Therefore, the agent is more rewarded by putting a wall than by putting a window on this column. In order to get better segmentations,



Figure 5.10: Parsing images from New York City, USA.

we present now some results with the 4-colors grammar (that contains the binary grammar) using a better suited type of image-based reward.

5.3.2 4-colors Facades

This grammar is very generic and most of the buildings world-wide can be explained by such a grammar. In the experiments, we combine it with a GMM reward, painting a few pixels in the original image to train the GMM. We first present parsing results on various buildings from New York City, USA in Fig.5.10.

In the "villages" in New York City, the appearance of the windows, walls and roof are efficiently captured by GMM since the colors are relatively uniform in each region. Therefore it is interesting to confront the parsing method with more challenging facade appearances, such as buildings from Barcelona, Spain, which is known world-wide to be very creative and diverse or Budapest, Hungary (see Fig.5.11).

All these facade segmentations have been obtained with a few brush strokes to learn a GMM classifier used as reward. A Q-learning agent learns for around 3000 episodes, using a data-driven exploration policy. Having a look at the building facades parsed so far, they all seem to share similar



Figure 5.11: Barcelona and Budapest Data Sets parsed using the 4-colors grammar.

topologies. The number of floors is always between 4 and 6 and the number of windows per floor around 5 as well. Since we claimed that the parsing algorithm was not making any assumption of the facades topologies and therefore could handle any kind of layouts, we now show some examples of parses on higher buildings from the USA (California and New York City) in Fig.5.12.

5.3.3 Haussmannian Architecture

The Haussmannian architecture gives an example of complex layouts, where the relationships between the different elements are more sophisticated than a mere grid. The grammar to describe it contains 7 terminal elements: wall, window, roof, shop, door, balcony and sky (it is the vocabulary of the Haussmannian grammar). Among them, some have a simple appearance model, while

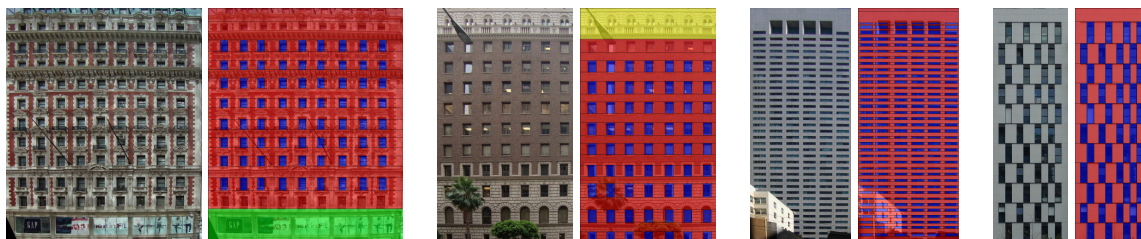


Figure 5.12: Higher Buildings.

others show a wider variety. Therefore, using either Randomized Forest rewards or Gaussian Mixture Models rewards will provide noisy information, which is usually enough to guide the learning agent toward the optimal parse.

We give now some examples of Haussmannian buildings from Paris, parsed with the Haussmann BSG in Fig.5.13. For these buildings, we let the agent learn for 5000 episodes. In this figure, we can see that even when the reward is noisy, the structure of the

5.3.4 Modern Architectures

Most of the buildings we have presented so far are built according to an underlying grid structure, which can be regular or irregular. It corresponds generally speaking to classical facade layouts. However, shape grammar can express more complex structures that correspond to more modern architecture (see Fig.5.12 on the right). To illustrate this statement, we extend the binary segmentation grammar by simply adding a second kind of floor. The two kinds of floor are alternating. Such buildings cannot be segmented by methods that assume a grid layout, since two grid layouts are tangled up. However, a shape grammar easily handles such a case, and parsing it is not more difficult than parsing any other grammar.

5.3.5 Robustness to Occlusions

The parsing algorithm is genuinely robust to occlusion. It inherits this property from the grammar and the definition of the state. Indeed, if for instance a window on a floor is occluded by an object, but that the windows below or above are not, then the agent considers that putting a window there is in average a good choice, and will therefore choose to put one. All the floors of the facade being treated as one, they are cooperating to help the agent choose the best way to split them. In practice, we observe that the grammar can handle pretty large occlusions, relying on the shape grammar to fill the blank in. In Fig.5.14, we study the reaction of the parsing algorithm to artificial occlusions. For that matter, we draw black rectangles on the image.

As we can see in Fig.5.14, the agent is not bothered by an occlusion on a floor. Of course, if all the windows of a column are occluded, there is no way the agent can recover them. However, he will recover the windows as long as it is more profitable than not putting them.

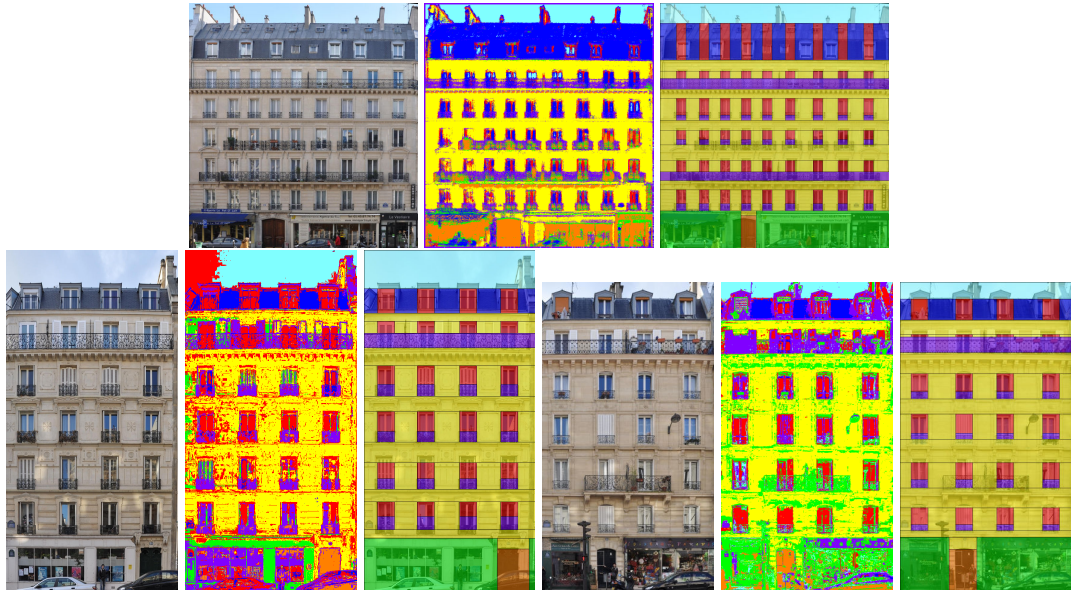


Figure 5.13: Parsing Haussmannian building with a Haussmannian BSG. Note that the topology of the optimal parse is always consistent with the expected topology. The errors on the appearance are usually very small, and are due to poorly informative data terms (rewards), as it can be observed in the MAP segmentations.

In Fig.5.15, we present some results of natural occlusions. In practice, trees, traffic lights, signalization, flags may hide the elements of the facade, making it harder for the learning agent to parse it. However, the constraints imposed by the grammar and by the agent allow him to recover the hidden structures.

Now that we have seen that the parsing algorithm is naturally robust to large occlusions, let's discuss the robustness to challenging illumination conditions.

5.3.6 Robustness to Illumination

Basically, two main problems main arise with illumination: cast shadows on the facade, or night lights. For these two problems there are two solutions. First, the difference of illumination can be partly solved by the reward. For instance, with a GMM reward, one can train the GMM with pixels in the sun or in the shade. Besides, if the reward fails at detecting regions of the image that receive particular illuminations, then these regions are treated as occlusions. These regions do not provide any meaningful information to the agent, but while parsing other regions he will be able to understand the way to tackle these uninformative regions.

Fig.5.16 shows the result of parsing facades that are partly in the sun and partly in the shade.

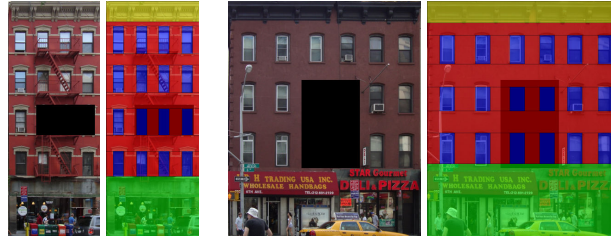


Figure 5.14: Robustness to artificial occlusions. Since all the floors of the facade are somehow treated as one by the agent, and since the agent is constrained by the shape grammar itself, an occlusion on a floor does not prevent him from splitting the floor correctly.

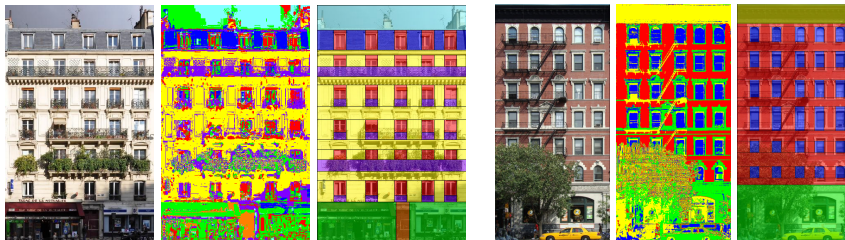


Figure 5.15: Robustness to natural occlusions. In these two examples, a tree and some vegetation are occluding a significant part of the facade. The data term misclassifies these pixels, but the parsing algorithm takes advantage of the global structure of the building to solve the occlusions.

For these cases, we can observe in the MAP image that the GMM reward actually takes into account the two different appearances of the same symbols.

Fig. 5.17 shows night pictures of Haussmannian buildings. Therefore, some pixels are saturated due to the presence of street lamp in front of the facade. The street lamp as well as the halo of light surrounding it completely loose the GMM classifier. Thus, in these cases, the challenging illumination is actually equivalent to an occlusion.

5.3.7 From 2D to 3D: Image-based Procedural Building Modeling

The gap from 2D to 3D is not as big as it is in general in Computer Vision while dealing with shape grammars. From the 2D structure of a building, it is very easy to infer a 3D model by using a 3D shape grammar presented in chapter 2. For that matter, we simply use the same rules as in 2D to generate a facade, and then add some grammar rules to intrude some symbols and extrude others. We end-up with a coarse 3D model on which we can directly map the input image as a texture for the facade.

However, procedural modeling is a much more powerful tool that allows us to go further. Since

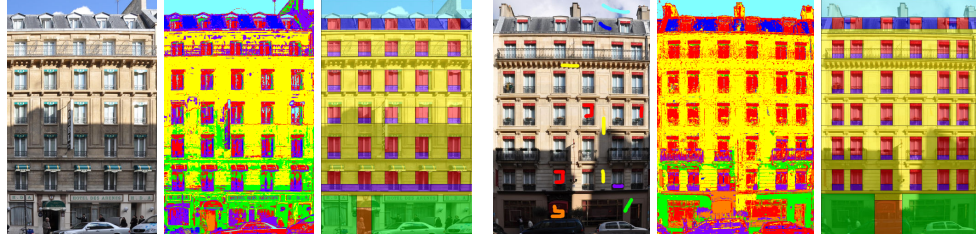


Figure 5.16: Robustness to cast shadows. Huge shadows are cast onto the facades. This is solved by the parsing algorithm both by taking it into account in the data term and treating the misclassified part of the facade as occlusions.



Figure 5.17: Robustness to night illumination. Since the GMM reward is sometimes too weak by itself, the user defined constraints helps the agent on to properly parse the input images.

the segmentation is semantic, we can associate some 3D models to some symbols (for instance windows, door and balconies), or enrich the model by adding specific rules. For example, it is straightforward to write a rule that adds balcony support below the running balconies, respecting the positions of the windows below, or to add a rule that creates cornices between the facade and the roof or one that make chimneys on the roof. Such 3D models cannot be qualified as 3D reconstructions, because the parameters of the intrusions and extrusions are not optimized but guessed. As a consequence the term of image-based modeling is better suited to this vision task. Note that these parameters can be set very accurately by studying the architectural styles themselves. For example, in Haussmannian architecture, the depth of the running balconies was limited by law to 80cm so as to avoid fires, at a time when making a fire in the street was common.

Fig.5.18 shows the different steps of building a full 3D model from a single view: the original image, the 2D parsing, a coarse texture mapped model, a model with handmade 3D models for specific elements, and eventually the complete model with additional rules. Note that going from one representation to the next one is almost free, since we just add very simple rules to the existing grammar. The real difficulty lies in bridging the semantic gap from the original image to the 2D parsing, not in bridging the gap between 2D and 3D.

5.4 Conclusion

In this chapter, we have presented several reward functions so as to extend the parsing algorithm presented in chapter 4 to real data. These reward function correspond to several types of data and several types of grammars. In practice, it provides very satisfying semantic segmentations for most of the images of facades, and reaches state-of-the-art performances on existing databases, and a new and more complete database we introduced here.

In this chapter though, we certainly did not use the most sophisticated merit functions. However this was done on purpose, in order to evaluate the quality of the parsing algorithm disregarding the quality of the data term (merit function). For sure, the better the rewards, the more efficient the parsing is. This parsing algorithm is both faster and more robust than the state-of-the-art algorithms in shape grammar parsing. Not only does it give better quantitative and qualitative performances but it also fits the procedural modeling paradigm.

The algorithm has been tested exhaustively on many images and was used for single-view image-based 3D procedural modeling of buildings which was the main objective of this doctoral work. Bridging the semantic gap was a much more challenging vision task than bridging the dimensionality gap from 2D to 3D.

Furthermore, applying the same idea, this parsing algorithm could be extended to a full 3D parsing. Provided a set of calibrated images, one could apply the same principles in 3D. The state definition would actually be the same than in 2D and therefore increasing the dimension of the shapes does not increase the dimensionality of the state space. The dimension of the problem grows linearly with the number of decisions (non-terminal symbols), no matter the dimension of the shapes themselves. This extension to 3D is let to future work, since we do believe that this is not a contribution as significant as the one we proposed by reformulating the shape grammar parsing problem.

Moreover, consistently with the subject of this thesis, the proposed parsing algorithm is restricted to facade parsing, which is a sub-problem of the generic image parsing problem. Going from a very complex problem (image parsing) to a more simple one (facade parsing) has revealed the power of a mathematical modeling framework: MDP, which was so far barely explored in our scientific community. We do believe that the very same framework could turn out to be very powerful to tackle the generic problem. This is definitely not a straightforward extension, as 3D shape grammar parsing could be, but Reinforcement Learning possesses assets than could help bridging the semantic gap in Computer Vision.



Figure 5.18: From 2D parsing to 3D models. On the first row, we display different level of 3D modeling. From left to right: symbolic model, model textured with the facade image, textured model with complex 3D models for some terminal symbols (windows, door, balconies), enhanced complete 3D model. On the second row, we show some other complete models. Once the facades have been parsed, obtaining these 3D models is costless and straightforward.

Chapter 6

Conclusion

The presented work stems from the two worlds of Computer Graphics and Computer Vision. The objective of this thesis was to combine them in order to solve a common problem: image-based procedural building modeling. The key idea is that the shape grammars, brought by the Computer Graphics community, are a Rosetta stone to decipher the language of architecture, which needs Computer Vision to be applied on images. From the Computer Graphics point of view, solving this challenging problem constitutes an elegant solution to automatic 3D content creation, bridging the geometric gap from 2D to 3D. For the Computer Vision community, such problem is in line with the most challenging vision task: the semantic gap.

6.1 Contributions

In order to reach this objective, we have traversed the whole image-based procedural modeling loop. We have first detailed how procedural modeling could be used in a purely graphical task: generating a wide variety of complex buildings with a shape grammar. We have explained the mechanisms of procedural modeling, its operators and the way it ties geometry with semantics on specific examples. Although this first step is conceptually simple, it hides a sophisticated implementation that turns out to be necessary. The procedural engine is important to generate 3D models, but it is not sufficient by itself; the key is facade parsing.

To perform facade parsing, we proposed two methods: bottom-up or top-down. The first one is based on grouping similar detected features on the image. This approach is quite similar with several methods in the literature and suffers from several limitations. First it is limited to irregular grid layouts and do not use the power of shape grammars as an input, but rather as an output. Besides, the method relies in an unsupervised way on the image data, and could sometimes be mistaken by challenging architectures, occlusions or viewing conditions. Nonetheless, the method performs well on a great variety of images, and the lack of strong constraints necessarily leads

sometimes to failure cases in a completely unsupervised method processing real and challenging data.

The second parsing algorithm is the main contribution of this thesis. It stems from two complementary parsing families of algorithms: Monte-Carlo based methods in image parsing and Dynamic Programming based in formal grammar parsing. These two families of methods are indeed two extreme cases of the same theory: Markov Decision Process (MDP). The main contribution of this thesis is therefore a change of viewpoint: parsing a 2D shape grammar is formulated as a semi MDP. This formulation brings a whole optimization framework called Reinforcement Learning (RL), with efficient algorithms such as Q-Learning. Reinforcement Learning brings indeed more than optimization algorithms; it also supports many features such as function approximations or data-driven exploration. The integration of these features into the parsing algorithm constitutes the second major contribution of this work. Another contribution is the binary split grammar (BSG) proposed to efficiently model the buildings and that better fits the MDP framework: a short sequence of complex rules is turned into a long sequence of very simple and mutually recursive ones. To apply the parsing algorithm on real data, we proposed several simple data terms or *rewards* based on supervised or unsupervised image measurements, with and without user interaction. In spite of the simplicity of these rewards, the algorithm achieves state-of-the-art results on real data and show empirical convergences on artificial and real data, with different complexities of shape grammars.

Eventually, a minor contribution of this thesis concerns the experimental validation. In order to test the algorithms, we have created and shared two challenging benchmarks of annotated images of Parisian facades, and we also share on-line rectified images of buildings from different cities that were already used by several research groups.

6.2 Applications

The first application of this parsing algorithm is the direct application to image-based procedural modeling. The 2D parses are turned into 3D models that can be textured mapped with the parsed facades and very easily enhanced with the incorporation of handmade 3D models, leading to highly detailed buildings. Thus the use of shape grammars to bridge the semantic gap on facade images also performs at no extra cost automatic 3D content creation from images.

Moreover, we consider that the most important impact of this work is to be found in the Computer Vision community. The Reinforcement Learning framework was so far barely explored. However, it turned out to perform very well on this specific task. RL can be seen as an intelligent stochastic alternative to Dynamic Programming. Even though we did not try to apply RL on other problems, we would not be surprised if it turned out to give good results on other vision tasks where DP methods are currently used, and especially on parsing ones. As we hoped at the beginning, the study of a simpler problem has brought an elegant solution that could be applied to the full image

parsing problem. We did not solve the complete image parsing problem, but we hope that this work could be another step towards a solution.

6.3 Future Work

As future work, we envision several possible paths. The two first ones are direct extensions of the proposed work, while the other ones need more investigation.

Combining top-down and bottom-up parsing methods to boost the parsing performances. In this thesis, we separated on purpose the two methods so as to better study the contributions of each one. However, we do believe that advance bottom-up cues such as the ones presented in chapter 3 would necessarily improve the performances of the top-down algorithm.

3D shape grammar parsing with Reinforcement Learning. Indeed, the parsing algorithm as presented can already support 3D parsing. The extension is theoretically straightforward, but needs a lot of data acquisition and implementation in practice. The idea would be to parse the points cloud obtained with a structure and motion algorithm with a 3D split grammar, applying the same principles and the same parsing algorithm. This would really be the first 3D shape grammar parsing work, since the method we propose here turns a 2D parsing into a 3D models.

Texture enhancement and synthesis would be in line with Computer Graphics concerns. We have shown in this thesis that parsing real images improved a lot the graphical quality of procedural models. However, using the input facade image as a texture is often not satisfying, since the input photograph is often corrupted by occlusions due to street lights, road signs, pedestrians, cars, traffic lights, trees, etc., and by illumination conditions. Therefore, a first extension in this trend would be to perform automatic inpainting, based on the likelihood of the optimal parse, in order to clean out the facade texture. Then, a second extension would be to perform texture synthesis of new procedural buildings. Starting from an example of a facade, the parsing algorithm would segment the facade and use the parameters of the parse and the shape grammar to generate a new facade with a different size. Then, the goal would be to paint the pixels based on the optimal parse so as to create a new realistic facade. This optimization procedure may also rely on Reinforcement Learning, using the same loop for learning the original facade and synthesizing the new one.

Reward Learning is a challenging learning task that would improve the parsing quality. Quite often, the optimal parse found by the parsing algorithm brings a better return than the expected ground truth. Therefore, the blame cannot be put on the optimization, but on the poor informative quality of the reward, and having a better data term would lead to the expected solution. Therefore, one could for example associate a weight to each symbol and multiply the merit of each symbol by its weight while computing the rewards. The goal would be to learn from a dataset the values

of these weights, so that the optimal solution coincides with the expected one. Other learning approaches could be considered as well.

Multi agents grammar selection concerns the RL optimization. In chapter 4, we proposed a simple way to perform model selection, that is to say grammar selection. To my mind, this question is important. How to parse a facade with different grammars, so as to find the one that best fit. Some part of Reinforcement Learning studies multi-agents learning, and this seems to be a promising approach to tackle efficient grammar selection in facade parsing.

Image parsing. The idea would be to apply Reinforcement Learning as described in this thesis for facade parsing, to other parsing tasks in Computer Vision. This work is indeed very challenging and far from being straightforward, but we do believe that it is worth investigating.

Shape grammar inference is the third at last problem defined by James Gips. In this thesis we proposed to tackle the second problem of shape grammar parsing, leaving aside the more challenging one of inference. The goal here is to learn a shape grammar from examples of shapes. Here again, solving this problem is far from being straightforward, and some attempts have been made in the literature. Such an algorithm would have an tremendous impact on the Computer Graphics community, since so far designing a shape grammar remains the weak point procedural modeling.

Publications of the Author

International Journals

- Loic Simon, Olivier Teboul, Panagiotis Koutsourakis and Nikos Paragios. *Random Exploration of the Procedural Space for Single-View 3D Modeling of Buildings*. International Journal of Computer Vision, pages 1–19, 2010.

International Conferences

- P Koutsourakis, L Simon, O Teboul, G Tziritas and N Paragios. *Single view reconstruction using shape grammars for urban environments*. In Computer Vision, 2009 IEEE 12th International Conference on. IEEE, 2009.
- Olivier Teboul, Loic Simon, Panagiotis Koutsourakis and Nikos Paragios. *Segmentation of building facades using procedural shape priors*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3105–3112, San Francisco, USA, 2010. IEEE. **Oral presentation.**
- Olivier Teboul, Iasonas Kokkinos, Panagiotis Koutsourakis, Loic Simon and Nikos Paragios. *Shape Grammar Parsing via Reinforcement Learning*. In Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, pages 3105–3112. IEEE, 2011.

Bibliography

- [Agarwal 1998] Maneesh Agarwal and Jonathan Cagan. *A blend of different tastes: the language of coffeemakers*. Environment and Planning B: Planning and Design, vol. 25, no. 2, pages 205–226, 1998.
- [Agarwal 2010] S Agarwal, Noah Snavely, I Simon, Steve Seitz and Richard Szeliski. *Building rome in a day*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 72–79. IEEE, 2010.
- [Aichholzer 1995] Oswin Aichholzer, Franz Aurenhammer, David Alberts and Bernd Gärtner. *A Novel Type of Skeleton for Polygons*. Journal of Universal Computer Science, vol. 1, no. 12, pages 752–761, 1995.
- [Alegre 2004] Fernando Alegre and Frank Dellaert. *A probabilistic approach to the semantic interpretation of building facades*. In International Workshop on Vision Techniques Applied to the Rehabilitation of City Centres, volume 2, page 3. Citeseer, 2004.
- [Ali 2007] Haider Ali, Christin Seifert, Nitin Jindal, Lucas Paletta and Gerhard Paar. *Window Detection in Facades*. International Conference on Image Analysis and Processing (ICIAP), pages 837–842, September 2007.
- [Aliaga 2008] Daniel Aliaga, Carlos Vanegas and Bedrich Benes. *Interactive example-based urban layout synthesis*. ACM Transactions on Graphics, vol. 27, no. 5, page 1, December 2008.
- [Arya 1998] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman and Angela Y Wu. *An optimal algorithm for approximate nearest neighbor searching fixed dimensions*. Journal of the ACM, vol. 45, no. 6, pages 891–923, 1998.
- [Baillard 1999] Caroline Baillard and Andrew Zisserman. *Automatic reconstruction of piecewise planar models from multiple views*. Proceedings 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Cat No PR00149, vol. 2, pages 559–565, 1999.

- [Barto 2003] Andrew Barto and Sridhar Mahadevan. *Recent advances in hierarchical reinforcement learning*. Discrete Event Dynamic Systems, vol. 13, no. 4, pages 341–379, 2003.
- [Bauer 2002] Joachim Bauer, Andreas Klaus, Konrad Karner, Christopher Zach and Konrad Schindler. *MetropoGIS: A feature based city modeling system*. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS), vol. 34, no. 3/B, pages 22–27, 2002.
- [Bay 2006] H Bay, T Tuytelaars and Luc Van Gool. *Surf: Speeded up robust features*. In European Conference on Computer Vision (ECCV), pages 404–417, Graz, 2006. Springer.
- [Bekins 2005] Daniel Bekins and Daniel Aliaga. *Build-by-number: Rearranging the real world to visualize novel architectural spaces*. In IEEE Visualization, pages 143–150. IEEE, 2005.
- [Bellman 1957a] Richard Bellman. *A Markovian Decision Process*. Journal of Mathematics and Mechanics, vol. 6, 1957.
- [Bellman 1957b] Richard Bellman. Dynamic Programming. Princeton University Press, 1957.
- [Berg 2007] Alexander C. Berg, Floraine Grabler and Jitendra Malik. *Parsing Images of Architectural Scenes*. IEEE International Conference on Computer Vision (ICCV), pages 1–8, 2007.
- [Bertsekas 2007] Dimitri Bertsekas. *Neuro-dynamic programming: An overview and recent results*. In Operations Research Proceedings 2006, pages 71–72. Springer, 2007.
- [Bishop 2006] Christopher Bishop. Pattern Recognition and Machine Learning. Springer, 2006.
- [Blake 2004] Andrew Blake, Carsten Rother, M Brown, Patrick Perez and Philip Torr. *Interactive image segmentation using an adaptive GMMRF model*. In European Conference on Computer Vision (ECCV), pages 428–441, 2004.
- [Boykov 2001] Yuri Boykov, O Veksler and Ramin Zabih. *Fast Approximate Energy Minimization via Graph Cuts*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 11, pages 1222–1239, 2001.
- [Breiman 2001] Leo Breiman. *Random Forests*. Machine Learning, vol. 45, no. 1, pages 5–32, 2001.
- [Brown 2005] M. Brown and D.G. Lowe. *Unsupervised 3D Object Recognition and Reconstruction in Unordered Datasets*. In International Conference on 3-D Digital Imaging and Modeling, pages 56–63. Ieee, 2005.
- [Burochin 2009] Jean-pascal Burochin, Olivier Tournaire and Nicolas Papanoditis. *An Unsupervised Hierarchical Segmentation of a Facade Building Image in Elementary 2D-Models*. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS), vol. 38, pages 223–228, 2009.

- [Cagdas 1996] G Cagdas. *A shape grammar: the language of traditional Turkish houses*. Environment and Planning B: Planning and Design, vol. 23, no. 4, pages 443–464, 1996.
- [Cantor 1883] Georg Cantor. *Über unendliche, lineare Punktmannigfaltigkeiten*. Mathematische Annalen, vol. 21, pages 545–591, 1883.
- [Cech 2008] Jan Cech and Radim Sara. *Windowpane detection based on maximum a posteriori probability labeling*. In International Workshop on Combinatorial Image Analysis, pages 3–11, Buffalo, NY, USA, 2008.
- [Cech 2009] Jan Cech and Radim Sara. *Languages for constrained binary segmentation based on maximum a posteriori probability labeling*. International Journal of Imaging Systems and Technology, vol. 19, no. 2, pages 69–79, June 2009.
- [Chen 2007a] Yuanhao Chen, Alan Yuille and Long Leo Zhu. *Unsupervised Learning of a Probabilistic Grammar for Object Detection and Parsing*. Science And Technology, pages 827–834, 2007.
- [Chen 2007b] Yuanhao Chen, Long Leo Zhu, Chenxi Lin, Alan Yuille and H Zhang. *Rapid inference on a novel and/or graph for object detection, segmentation and parsing*. Advances in Neural Information Processing Systems (NIPS), pages 1–8, 2007.
- [Chen 2008] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller and Eugene Zhang. *Interactive procedural street modeling*. In ACM Transactions on Graphics, pages 1–10. ACM, 2008.
- [Choi 1997] Hyeong In Choi, Sung Woo Choi and Hwan Pyo Moon. *Mathematical theory of medial axis transform*. Pacific Journal of Mathematics, vol. 181, no. 1, pages 57–88, November 1997.
- [Chomsky 1956] Noam Chomsky. *Three Models for the description of Language*. IRE Transactions on Information Theory, vol. 2, pages 113–123, 1956.
- [Coelho 2007] A. Coelho, M. Bessa, A.A. Sousa and Fernando Nunes Ferreira. *Expeditious modelling of virtual urban environments with geospatial L-systems*. In Computer Graphics Forum, volume 26, pages 769–782. John Wiley & Sons, 2007.
- [Comaniciu 2002] Dorin Comaniciu and Peter Meer. *Mean Shift: A Robust Approach Toward Feature Space Analysis*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pages 603–619, 2002.
- [Cornelis 2006] Nico Cornelis, Bastian Leibe, Kurt Cornelis and Luc Van Gool. *3D City Modeling Using Cognitive Loops*. International Symposium on Virtual Reality, Archeology and Virtual Heritage (VAST), pages 9–16, June 2006.

- [Debevec 1996] Paul Debevec, Christopher Taylor and Jitendra Malik. *Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach*. In Annual Conference on Computer Graphics and Interactive Techniques, page 20. ACM, 1996.
- [Dick 2000] Anthony Dick, Philip Torr and Roberto Cipolla. *Automatic 3d modelling of architecture*. In British Machine Vision Conference (BMVC), pages 273–289. Citeseer, 2000.
- [Dick 2002] Anthony Dick, Philip Torr and Roberto Cipolla. *A bayesian estimation of building shape using MCMC*. In European Conference on Computer Vision (ECCV), pages 574–575. Springer, 2002.
- [Dietterich 2000] T G Dietterich. *Hierarchical reinforcement learning with MAXQ*. Journal of Artificial Intelligence Research, vol. 13, 2000.
- [Dylla 2009] Kimberly Dylla, Bernard Frischer, Pascal Müller, Andreas Ulmer and Simon Hae-gler. *Rome Reborn 2.0: A Case Study of Virtual City Reconstruction Using Procedural Modeling Techniques*. Annual Conference of the College Art Association, vol. 16, 2009.
- [Earley 1970] Jay Earley. *An efficient context-free parsing algorithm*. Communication of the ACM, vol. 13, no. 2, 1970.
- [Eppstein 1999] David Eppstein and Jeff Erickson. *Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions*. Discrete & Computational Geometry, vol. 22, no. 4, pages 569–592, 1999.
- [Faugeras 1993] Olivier Faugeras. *Three-dimensional computer vision: a geometric viewpoint*. MIT Press, 1993.
- [Felzenszwalb 2007] Pedro F Felzenszwalb and David Mcallester. *The Generalized A * Architecture*. Artificial Intelligence, vol. 29, pages 153–190, 2007.
- [Felzenszwalb 2010a] Pedro F Felzenszwalb and Ramin Zabih. *Dynamic Programming and Graph Algorithms in Computer Vision*. IEEE transactions on pattern analysis and machine intelligence, pages 1–51, July 2010.
- [Felzenszwalb 2010b] P.F. Felzenszwalb and Olga Veksler. *Tiered scene labeling with dynamic programming*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3097–3104, San Francisco, USA, 2010. IEEE.
- [Finkenzeller 2006] Dieter Finkenzeller and Alfred Schmitt. *Representation of complex façades using typed graphs*. In Virtual Concept, pages 1–8, Playa del Carmen, Mexico, 2006. Citeseer.
- [Finkenzeller 2008a] D Finkenzeller and A Schmitt. *Rapid modeling of complex building façades*, 2008.

- [Finkenzeller 2008b] Dieter Finkenzeller. *Detailed Building Facades*. IEEE Computer Graphics and Applications, vol. 28, no. 3, pages 58–66, May 2008.
- [Fischler 1981] M.A. Fischler and R.C. Bolles. *Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography*. Communications of the ACM, vol. 24, no. 6, pages 381–395, 1981.
- [Flemming 1987] U Flemming. *More than the sum of parts: the grammar of Queen Anne houses*. Environment and Planning B: Planning and Design, vol. 14, no. 3, pages 323–350, 1987.
- [Frahm 2009] Jan-Michael Frahm, Marc Pollefeys, Brian Clipp, David Gallup, Rahul Raguram, C.C. Wu and Christopher Zach. *3d reconstruction of architectural scenes from uncalibrated video sequences*. 3D Virtual Reconstruction and Visualization of Complex Architectures, vol. 38, no. 5/W1, page 7, 2009.
- [Fraundorfer 2006] Friedrich Fraundorfer, Konrad Schindler and Horst Bischof. *Piecewise planar scene reconstruction from sparse correspondences*. Image and Vision Computing, vol. 24, no. 4, pages 395–406, 2006.
- [Furukawa 2010] Yasutaka Furukawa and Jean Ponce. *Accurate, dense, and robust multiview stereopsis*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 8, pages 1362–76, 2010.
- [Gallup 2010] David Gallup, Jan-Michael Frahm and Marc Pollefeys. *Piecewise planar and non-planar stereo for urban scene reconstruction*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1418–1425, San Francisco, USA, 2010. IEEE.
- [Ganster 2007] B. Ganster and Reinhard Klein. *An integrated framework for procedural modeling*. In Spring Conference on Computer Graphics (SCCG), volume 7, pages 150–157. Citeseer, 2007.
- [Gips 1999] James Gips. *Computer implementation of shape grammars*. In NSF/MIT Workshop on Shape Computation. Citeseer, 1999.
- [Greuter 2003] Stefan Greuter, Jeremy Parker, Nigel Stewart and Geoff Leach. *Real-time procedural generation of pseudo infinite cities*. In International conference on Computer graphics and interactive techniques in Australasia and South East Asia, pages 87–95, New York, New York, USA, 2003. Citeseer.
- [Harris 1988] C Harris and M. Stephens. *A combined corner and edge detector*. In Alvey vision conference, volume 15, page 50. Manchester, UK, 1988.
- [Hartley 1992] Richard I Hartley, Rajiv Gupta and Tom Chang. *Stereo from uncalibrated cameras*. IEEE Computer Vision and Pattern Recognition (CVPR), pages 761–764, 1992.

- [Hartley 2003] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, second édition, 2003.
- [Havemann 2001] Sven Havemann and Dieter Fellner. *A versatile 3D model representation for cultural reconstruction*. International Symposium on Virtual Reality, Archeology and Cultural Heritage (VAST), page 205, 2001.
- [Havemann 2004] S. Havemann and D. Fellner. *Generative parametric design of Gothic window tracery*. Proceedings Shape Modeling Applications, pages 350–353, 2004.
- [Hays 2006] James Hays, Marius Leordeanu, Alexei Alyosha Efros and Yanxi Liu. *Discovering texture regularity as a higher-order correspondence problem*. In European Conference on Computer Vision (ECCV), pages 522–535. Springer, 2006.
- [Hengel 2006] A Hengel, Anthony Dick, T Thormahlen, Ben Ward and Philip Torr. *Rapid interactive modelling from video with graph cuts*. In Eurographics, 2006.
- [Hernández 2009] Jorge Hernández and Beatriz Marcotegui. *Morphological segmentation of building façade images*. In IEEE International Conference on Image Processing (ICIP), pages 4029–4032, Cairo, Egypt, 2009. IEEE.
- [Hiep 2009] V.H. Hiep, Renaud Keriven, Patrick Labatut and Jean-Philippe Pons. *Towards high-resolution large-scale multi-view stereo*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1430–1437, June 2009.
- [Howard 1960] Ronald Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
- [Kadir 2004] Timor Kadir, Andrew Zisserman and Michael Brady. *An affine invariant salient region detector*. European Conference on Computer Vision (ECCV), pages 228–241, 2004.
- [Kiryati 1998] Nahum Kiryati and Yossi Gofman. *Detecting symmetry in grey level images: The global optimization approach*. International Journal of Computer Vision, 1998.
- [Klein 1994] Dan Klein and Christopher D Manning. *A * Parsing : Fast Exact Viterbi Parse Selection*. Science, 1994.
- [Knight 1981] T Weissman Knight. *The forty-one steps*. Environment and Planning B: Planning and Design, vol. 8, no. 1981, pages 97–114, 1981.
- [Knight 1994] T Weissman Knight. *Transformations in Design: a Formal Approach to Stylistic Change and Innovation in the Visual Arts*. Cambridge édition, 1994.
- [Kokkinos 2008] Iasonas Kokkinos and Alan Yuille. *Scale invariance without scale selection*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8. IEEE, June 2008.

- [Kokkinos 2009] Iasonas Kokkinos and Alan Yuille. *Hop: Hierarchical object parsing*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), volume 1, Miami, USA, 2009. IEEE.
- [Kokkinos 2010] Iasonas Kokkinos. *Boundary Detection using F-Measure- , Filter- and*. In European Conference on Computer Vision (ECCV), Heraklio, Greece, 2010.
- [Komodakis 2009] Nikos Komodakis, Nikos Paragios and Giorgios Tziritas. *Clustering via LP-based stabilities*. In Advances in neural information processing systems (NIPS), 2009.
- [Korah 2006] Thommen Korah and Christopher Rasmussen. *Improving spatiotemporal inpainting with layer appearance models*. Advances in Visual Computing, pages 718–730, 2006.
- [Korah 2008] Thommen Korah and Christopher Rasmussen. *Analysis of building textures for reconstructing partially occluded facades*. In European Conference on Computer Vision (ECCV), pages 359–372, Heraklio, Greece, 2008. Springer Berlin/Heidelberg.
- [Koutsourakis 2009a] P Koutsourakis, L Simon, O Teboul, G Tziritas and N Paragios. *Single view reconstruction using shape grammars for urban environments*. In Computer Vision, 2009 IEEE 12th International Conference on. IEEE, 2009.
- [Koutsourakis 2009b] Panagiotis Koutsourakis, Loic Simon, Olivier Teboul, Giorgios Tziritas and Nikos Paragios. *Single View Reconstruction Using Shape Grammars for Urban Environments*. In IEEE International Conference on Computer Vision (ICCV), 2009.
- [Lafarge 2010] Florent Lafarge, Renaud Keriven, M. Brédif and V.H. Hiep. *Hybrid multi-view reconstruction by Jump-Diffusion*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 350–357. IEEE, 2010.
- [Lee 2004] Sung Chun Lee and Ram Nevatia. *Extraction and Integration of Window in a 3D Building Model from Ground View images*. In IEEE Conference on Computer Vision and Pattern Recognition, volume 1, 2004.
- [Lepetit 2006] Vincent Lepetit and Pascal Fua. *Keypoint Recognition Using Randomized Trees*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 9, pages 1465–1479, 2006.
- [Leung 1996] Thomas Leung and Jitendra Malik. *Detecting, localizing and grouping repeated scene elements from an image*. European Conference on Computer Vision (ECCV), no. April, pages 546–555, 1996.
- [Levinshtein 2009] Alex Levinshtein, Adrian Stere, Kiriakos N Kutulakos, David J Fleet, Sven J Dickinson and Kaleem Siddiqi. *TurboPixels: fast superpixels using geometric flows*. IEEE transactions on pattern analysis and machine intelligence, vol. 31, no. 12, pages 2290–7, December 2009.

- [Li 2008] Lihong Li and Michael Littman. *Prioritized Sweeping Converges to the Optimal Value Function*, 2008.
- [Lindeberg 1998] Tony Lindeberg. *Feature detection with automatic scale selection*. International Journal of Computer Vision, vol. 30, no. 2, pages 79–116, 1998.
- [Lindenmayer 1968] Aristid Lindenmayer. *Mathematical Models of cellular interaction in development, {P}arts {I} and {II}*. Journal of Theoretical Biology, vol. 18, pages 280–315, 1968.
- [Lipp 2008] Markus Lipp, Peter Wonka and Michael Wimmer. *Interactive Visual Editing of Grammars for Procedural Architecture*. ACM Transactions on Graphics, vol. 27, no. 3, pages 102:1—10, 2008.
- [Liu 2004] Yanxi Liu, Robert T Collins and Yanghai Tsin. *A computational model for periodic pattern perception based on frieze and wallpaper groups*. IEEE transactions on pattern analysis and machine intelligence, vol. 26, no. 3, pages 354–71, March 2004.
- [Liu 2005] Y. Liu, Y. Tsin and W.C. Lin. *The promise and perils of near-regular texture*. International Journal of Computer Vision, vol. 62, no. 1, pages 145–159, 2005.
- [Liu 2010] Chun Liu and André Gagalowicz. *Image-based Modeling of Hausmannian Facades*. International Journal of Virtual Reality, vol. 9, no. 1, pages 13–18, 2010.
- [Lourakis 2004] M I A Lourakis and A A Argyros. *The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm*. ICSFORTH Technical Report TR, vol. 340, no. 340, 2004.
- [Lowe 1999] David Lowe. *Object Recognition from Local Scale-Invariant Features*. In IEEE International Conference on Computer Vision (ICCV), 1999.
- [Lowe 2004] David Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. International Journal of Computer Vision, vol. 60, no. 2, 2004.
- [Loy 2006] Gareth Loy and J.O. Eklundh. *Detecting symmetry and symmetric constellations of features*. European Conference on Computer Vision (ECCV), pages 508–521, 2006.
- [Loyer 1987] François Loyer. *Paris XIXe siecle : l'immeuble et la rue*, Paris. Hazan édition, 1987.
- [Lucas 1981] Bruce D. Lucas and Takeo Kanade. *An Iterative Image Registration Technique with an Application to Stereo Vision*. , pages , 1981. In International Joint Conference on Artificial Intelligence, pages 674–679, 1981.
- [Martin 2004] D.R. Martin, C.C. Fowlkes and Jitendra Malik. *Learning to detect natural image boundaries using local brightness, color, and texture cues*. IEEE transactions on pattern analysis and machine intelligence, vol. 26, no. 5, pages 530–49, May 2004.

- [Marvie 2005] Jean-Eudes Marvie, Julien Perret and Kadi Bouatouch. *The FL-system: a functional L-system for procedural geometric modeling*. The Visual Computer, vol. 21, no. 5, pages 329–339, May 2005.
- [Matas 2004] J Matas, O. Chum, M. Urban and T. Pajdla. *Robust wide-baseline stereo from maximally stable extremal regions*. Image and Vision Computing, vol. 22, no. 10, pages 761–767, September 2004.
- [McCormack 2004] Jay P McCormack, Jonathan Cagan and Craig M Vogel. *Speaking the Buick language: capturing, understanding, and exploring brand identity with shape grammars*. Design Studies, vol. 25, no. 1, pages 1–29, January 2004.
- [Mikolajczyk 2004] Krystian Mikolajczyk and Cordelia Schmid. *Scale and Affine Invariant Interest Point Detectors*. International Journal of Computer Vision, vol. 60, no. 1, 2004.
- [Mitra 2006] Niloy Mitra, Leonidas Guibas and Mark Pauly. *Partial and approximate symmetry detection for 3D geometry*. ACM Transactions on Graphics, vol. 25, no. 3, pages 560–568, July 2006.
- [Mitra 2008] Niloy Mitra and Mark Pauly. *Symmetry for architectural design*. Advances in Architectural Geometry, pages 13–16, 2008.
- [Moore 2008] Alastair P. Moore, Simon J. D. Prince, Jonathan Warrell, Umar Mohammed and Graham Jones. *Supapixel lattices*. 2008 IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8, June 2008.
- [Mordohai 2007] Philippos Mordohai, Jan-Michael Frahm, A Akbarzadeh, Brian Clipp, C Engels, David Gallup, Paul Merrell, C Salmi, Sudipta Sinha, B Talton, L Wang, Q Yang, H Stewenius, H Towles, G Welch, R Yang, Marc Pollefeys and D Nister. *Real-time video-based reconstruction of urban environments*. In 3D Virtual Reconstruction and Visualization of Complex Architectures. Citeseer, 2007.
- [Morel 2009] Jean-Michel Morel and Guoshen Yu. *ASIFT: A New Framework for Fully Affine Invariant Image Comparison*. SIAM Journal on Imaging Sciences, vol. 2, no. 2, page 438, 2009.
- [Mori 2004] G. Mori, A.a. Efros and J. Malik. *Recovering human body configurations: combining segmentation and recognition*. In IEEE Conference on Computer Vision and Pattern Recognition, numéro c, pages 326–333. Ieee, 2004.
- [Mudge 2005] M. Mudge, N. Ryan and R. Scopigno. *3D Modeling for Non-Expert Users with the Castle Construction Kit v0. 5*. In M. Mudge, N. Ryan and R. Scopigno, editeurs, International Symposium on Virtual Reality, Archeology and Cultural Heritage (VAST), pages 1–9. Citeseer, 2005.

- [Müller] Pascal Müller. *www.procedural.com*.
- [Müller 2005] Pascal Müller, T. Vereenooghe, Andreas Ulmer and Luc Van Gool. *Automatic reconstruction of Roman housing architecture*. International Workshop on Recording, Modeling and Visualization of Cultural Heritage, pages 287–297, 2005.
- [Müller 2006a] Pascal Müller, T. Vereenooghe, Peter Wonka, I Paap and Luc Van Gool. *Procedural 3D Reconstruction of Puuc Buildings in Xkipché*. International Symposium on Virtual Reality, Archeology and Cultural Heritage (VAST), pages 139–146, 2006.
- [Müller 2006b] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer and Luc Van Gool. *Procedural modeling of buildings*. ACM Transactions on Graphics, vol. 25, no. 3, page 614, July 2006.
- [Müller 2007] Pascal Müller, G. Zeng, Peter Wonka and Luc Van Gool. *Image-based procedural modeling of facades*. ACM Transactions on Graphics, vol. 26, no. 3, page 85, 2007.
- [Musialski 2009] Przemyslaw Musialski, Peter Wonka, M Recheis, S Maierhofer and W Purgathofer. *Symmetry-based facade repair*. In Vision, Modeling, and Visualization Workshop (VMV). Citeseer, 2009.
- [Neu 2009] G Neu and C Szepesvári. *Training parsers by inverse reinforcement learning*. Machine Learning, 2009.
- [Oh 2001] Byong Mok Oh, Max Chen, Julie Dorsey and Frédo Durand. *Image-based modeling and photo editing*. ACM Transactions on Graphics, pages 433–442, 2001.
- [Ohta 1978] Y. Ohta, T. Kanade and T. Sakai. *An analysis system for scenes containing objects with substructures*. In International Joint Conference on Pattern Recognitions, volume 1, 1978.
- [Parish 2001] Yoav I H Parish and Pascal Müller. *Procedural modeling of cities*. In ACM Transactions on Graphics, pages 301–308, 2001.
- [Park 2008] Minwoo Park, R. Collins and Yanxi Liu. *Deformed lattice discovery via efficient mean-shift belief propagation*. In European Conference on Computer Vision (ECCV), pages 474–485, Marseille, France, 2008. Springer.
- [Park 2010] Minwoo Park, Kyle Brocklehurst, Robert T Collins and Yanxi Liu. *Translation-Symmetry-based Perceptual Grouping with Applications to Urban Scenes*. In Asian Conference on Computer Vision, Queenstown, New Zealand, 2010.
- [Pauly 2008] Mark Pauly, Niloy Mitra, Johannes Wallner, Helmut Pottmann and Leonidas Guibas. *Discovering Structural Regularity in {3D} Geometry*. ACM Transactions on Graphics, vol. 27, no. 3, pages #43, 1–11, 2008.

- [Pollefeys 1998] M. Pollefeys, R. Koch, M. Vergauwen and L. Van Gool. *Metric 3D surface reconstruction from uncalibrated image sequences*. In European Workshop SMILE, pages 139–154. Springer, 1998.
- [Pollefeys 2001] Marc Pollefeys, Luc Van Gool, Maarten Vergauwen, Kurt Cornelis, Frank Verbiest and Jan Tops. *Image-based 3D acquisition of archaeological heritage and applications*. International Symposium on Virtual Reality, Archeology and Cultural Heritage (VAST), page 255, 2001.
- [Pollefeys 2002] Marc Pollefeys and Luc Van Gool. *Visual modelling: from images to images*. The Journal of Visualization and Computer Animation, vol. 13, no. 4, pages 199–209, September 2002.
- [Prunsinkiewicz 1996] Przemyslaw Prunsinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1996.
- [Pugliese 2002] Michael Pugliese and Jonathan Cagan. *Capturing a rebel: modelling the Harley-Davidson brand through a motorcycle shape grammar*. Research in Engineering Design, vol. 13, pages 139–156, 2002.
- [Recky 2010] Michal Recky and Franz Leberl. *Windows Detection Using K-means in CIE-Lab Color Space*. IEEE International Conference on Pattern Recognition (ICPR), pages 356–359, August 2010.
- [Reznik 2007] S Reznik and H Mayer. *Implicit Shape Models, Model Selection, and Plane Sweeping for 3D Facade Interpretation*. In Photogrammetric Image Analysis (PIA), page 173, 2007.
- [Ripperda 2007] Nora Ripperda and Claus Brenner. *Data Driven Rule Proposal for Grammar Based Facade Reconstruction*. In Photogrammetric Image Analysis (PIA), page 1, 2007.
- [Ripperda 2009] Nora Ripperda and Claus Brenner. *Application of a formal grammar to facade reconstruction in semiautomatic and automatic environments*. In Proceedings of the 12 th AGILE Conference on GIScience, Hanover, Germany, pages 1–12, 2009.
- [Rosten 2009] Edward Rosten, R Porter and T Drummond. *Faster and better: A machine learning approach to corner detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2009.
- [Schaffalitzky 1999] F Schaffalitzky and A Zisserman. *Geometric grouping of repeated elements within images*. Shape, Contour and Grouping in Computer Vision, pages 81–81, 1999.
- [Schindler 2008] Grant Schindler, Panchapagesan Krishnamurthy, Roberto Lubliner and Frank Dellaert. *Detecting and matching repeated patterns for automatic geo-tagging in urban environments*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1–7, Anchorage, USA, June 2008. Ieee.

- [Schlecht 2007] Joseph Schlecht, Kobus Barnard, Ekaterina Spriggs and Barry Pryor. *Inferring Grammar-based Structure Models from 3D Microscopy Data*. 2007 IEEE Conference on Computer Vision and Pattern Recognition, no. June, pages 1–8, June 2007.
- [Scognamillo 2003] Renata Scognamillo, Gillian Rhodes, Concetta Morrone and David Burr. *A feature-based model of symmetry detection*. Proceedings. Biological sciences / The Royal Society, vol. 270, no. 1525, pages 1727–33, August 2003.
- [Seitz 2006] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein and Richard Szeliski. *A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms*. IEEE Conference on Computer Vision and Pattern Recognition, pages 519–528, 2006.
- [Shotton 2007] Jamie Shotton, John Winn, Carsten Rother and Antonio Criminisi. *TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context*. International Journal of Computer Vision, vol. 81, no. 1, pages 2–23, December 2007.
- [Shotton 2008] Jamie Shotton, Matthew Johnson and Roberto Cipolla. *Semantic texton forests for image categorization and segmentation*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2008.
- [Simon 2010] Loic Simon, Olivier Teboul, Panagiotis Koutsourakis and Nikos Paragios. *Random Exploration of the Procedural Space for Single-View 3D Modeling of Buildings*. International Journal of Computer Vision, pages 1–19, 2010.
- [Sinha 2006] S.N. Sinha, J.M. Frahm, M. Pollefeys and Y. Genc. *GPU-based video feature tracking and matching*. In EDGE, Workshop on Edge Computing Using New Commodity Architectures, volume 278. Citeseer, 2006.
- [Sinha 2008] Sudipta Sinha, Drew Steedly, Richard Szeliski, Maneesh Agrawala and Marc Pollefeys. *Interactive 3D architectural modeling from unordered photo collections*. ACM Transactions on Graphics, vol. 27, no. 5, pages 1–10, December 2008.
- [Sinha 2009] Sudipta Sinha, Drew Steedly and Richard Szeliski. *Piecewise planar stereo for image-based rendering*. In IEEE International Conference on Computer Vision (ICCV), Kyoto, Japan, 2009.
- [Snavely 2006] Noah Snavely, Steve Seitz and Richard Szeliski. *Photo tourism: exploring photo collections in 3D*. In ACM Transactions on Graphics, numéro c, pages 835–846. ACM, 2006.
- [Snavely 2008] Noah Snavely, Steve Seitz and Richard Szeliski. *Modeling the World from {Internet} Photo Collections*. International Journal of Computer Vision, vol. 80, no. 2, pages 189–210, 2008.

- [Stiny 1972] George Stiny and James Gips. *Shape Grammars and the Generative Specification of Painting and Sculpture*. In C V Freiman, editeur, *Information Processing 71*, volume 71, pages 1460–1465. North-Holland, 1972.
- [Stiny 1978] George Stiny and W J Mitchell. *The Palladian grammar*. *Environment and Planning B: Planning and Design*, vol. 5, pages 5–18, 1978.
- [Stiny 1980] George Stiny. *Introduction to shape and shape grammars*. *Environment and Planning B: Planning and Design*, vol. 7, pages 343–351, 1980.
- [Stiny 1982] George Stiny. *Spatial Relations and Grammars*. *Environment and Planning B: Planning and Design*, vol. 9, pages 113–114, 1982.
- [Strecha 2010] Christoph Strecha, Alexander M Bronstein, Michael M Bronstein and Pascal Fua. *Technical Report LDAHash : Improved matching with smaller descriptors*, 2010.
- [Sutton 1998] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. *IEEE Transactions on Neural Networks*, vol. 9, no. 5, page 1054, 1998.
- [Teboul 2010] Olivier Teboul, Loic Simon, Panagiotis Koutsourakis and Nikos Paragios. *Segmentation of building facades using procedural shape priors*. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3105–3112, San Francisco, USA, 2010. IEEE.
- [Teboul 2011] Olivier Teboul, Iasonas Kokkinos, Panagiotis Koutsourakis, Loic Simon and Nikos Paragios. *Shape Grammar Parsing via Reinforcement Learning*. In *Computer Vision and Pattern Recognition (CVPR)*, 2010 IEEE Conference on, pages 3105–3112. IEEE, 2011.
- [Teoh 2009] Soon Tee Teoh. *Generalized Descriptions for the Procedural Modeling of Ancient East Asian Buildings*. *Computational Aesthetics in Graphics, Visualization, and Imaging*, 2009.
- [Tesauro 1995] Gerald Tesauro. *Temporal Difference Learning and TD-Gammon*. *Communications of the ACM*, vol. 38, no. 3, 1995.
- [Tighe 2010] Joseph Tighe and Svetlana Lazebnik. *SuperParsing: Scalable Nonparametric Image Parsing with Superpixels*. In *European Conference on Computer Vision (ECCV)*, pages 352–365, Heraklio, Greece, 2010. Springer.
- [Tola 2010] Engin Tola, Vincent Lepetit and Pascal Fua. *DAISY: an efficient dense descriptor applied to wide-baseline stereo*. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 5, pages 815–830, 2010.
- [Tu 2002] Z W Tu and Song-Chun Zhu. *Image segmentation by data-driven Markov chain Monte Carlo*. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pages 657–673, 2002.

- [Turina 2001] a. Turina, T. Tuytelaars and L. Van Gool. *Efficient grouping under perspective skew*. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, no. Cvpr, pages I–247–I–254, 2001.
- [Van Den Hengel 2007] A. Van Den Hengel, Anthony Dick, T Thormahlen, Ben Ward and Philip Torr. *VideoTrace: rapid interactive scene modelling from video*. In ACM Transactions on Graphics, page 86. ACM, 2007.
- [Van Meerbergen 2002] G. Van Meerbergen, Maarten Vergauwen, Marc Pollefeys and Luc Van Gool. *A hierarchical symmetric stereo algorithm using dynamic programming*. International Journal of Computer Vision, vol. 47, no. 1, pages 275–285, 2002.
- [Vanegas 2010] Carlos Vanegas, Daniel Aliaga and Bedrich Benes. *Building Reconstruction using Manhattan-World Grammars*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), San Francisco, USA, 2010.
- [Vedaldi 2008] Andrea Vedaldi and B Fulkerson. *{VLFeat}: An Open and Portable Library of Computer Vision Algorithms*. `\url{http://www.vlfeat.org/}`, 2008.
- [von. Koch 1904] H von. Koch. *Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire*. Archiv för Matemat., vol. 1, pages 681–702, 1904.
- [Watkins 1989] Christopher Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.
- [Watkins 1992] Christopher Watkins and Peter Dayan. *Q-learning*. Machine Learning, vol. 8, no. 3, pages 279–292, 1992.
- [Weber 2009] Basil Weber, Pascal Müller, Peter Wonka and Markus Gross. *Interactive Geometric Simulation of 4D Cities*. Computer Graphics Forum, vol. 28, no. 2, pages 481–492, 2009.
- [Wendel 2010] Andreas Wendel, Michael Donoser and Horst Bischof. *Unsupervised Facade Segmentation using Repetitive Patterns*. In Annual Symposium of the German Association for Pattern Recognition (DAGM), pages 51–60, Darmstadt, Germany, 2010. Springer.
- [Wenzel 2008] S. Wenzel, M. Drauschke and W. Förstner. *Detection of repeated structures in facade images*. Pattern Recognition and Image Analysis, vol. 18, no. 3, pages 406–411, September 2008.
- [Whiting 2009] Emily Whiting, John Ochsendorf and Frédo Durand. *Procedural modeling of structurally-sound masonry buildings*. ACM Transactions on Graphics, vol. 28, no. 5, page 1, December 2009.
- [Winder 2007] Simon a. J. Winder and Matthew Brown. *Learning Local Image Descriptors*. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8. Ieee, June 2007.

- [Wonka 2003] Peter Wonka, Michael Wimmer, François X Sillion and William Ribarsky. *Instant architecture*. ACM Transactions on Graphics, vol. 22, no. 3, pages 669–677, 2003.
- [Wu 2010] Changchang Wu, JM Frahm and M. Pollefeys. *Detecting large repetitive structures with salient boundaries*. In European Conference on Computer Vision (ECCV), pages 142–155. Springer, 2010.
- [Wu 2011] Changchang Wu, J.M. Frahm and Marc Pollefeys. *Repetition-based Dense Single-View Reconstruction*. In IEEE Conference on Computer Vision and Pattern Recognition, 2011.
- [Yakubenko 2010] Anton Yakubenko, Ivan Mizin and Anton Konushin. *Automatic Extraction of Regular Grids from Rectified Facade Image*. In GraphIcon, 2010.
- [Younger 1967] Daniel H Younger. *Recognition and parsing of context-free languages in time n3*. Information and Control, vol. 10, no. 2, pages 189–208, 1967.
- [Zelevnik 2006] R.C. Zelevnik, K.P. Herndon and J.F. Hughes. *SKETCH: an interface for sketching 3D scenes*. In ACM SIGGRAPH Courses, page 9. ACM, 2006.
- [Zhan 2003] Yiqiang Zhan and Dinggang Shen. *Automated segmentation of 3D US prostate images using statistical texture-based matching method*. Medical Image Computing and Computer-Assisted Intervention, pages 688–696, 2003.
- [Zhu 2000] Song-Chun Zhu, R Zhang and Z W Tu. *Integrating Top-Down/Bottom-Up for Object Recognition by Data-Driven Markov Chain Monte Carlo*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2000.
- [Zhu 2006] Song-Chun Zhu and David Mumford. *A Stochastic Grammar of Images*. Foundations and Trends in Computer Graphics and Vision, vol. 2, no. 4, pages 259–362, 2006.
- [Zhu 2007] Song-Chun Zhu and David Mumford. *Quest for a Stochastic Grammar of Images*. Foundations and Trends in Computer Graphics and Vision, 2007.
- [Zhu 2008] Long Leo Zhu, Yuanhao Chen, Yuan Lin, Chenxi Lin and Alan Yuille. *Recursive Segmentation and Recognition Templates for 2D Parsing*. In Advances in neural information processing systems (NIPS), 2008.

