Analysis and Synthesis of Structured Variations in 3D Geometries

THÈSE N^{O} 6940 (2016)

PRÉSENTÉE LE 17 MARS 2016

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE D'INFORMATIQUE GRAPHIQUE ET GÉOMÉTRIQUE
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Ngoc Minh DANG

acceptée sur proposition du jury:

Prof. P. Dillenbourg, président du jury
Prof. M. Pauly, directeur de thèse
Prof. P. Wonka, rapporteur
Prof. M. Wand, rapporteur
Prof. S. Süsstrunk, rapporteuse

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2016

To my family.

Acknowledgements

First of all, I would like to express my sincere thanks to my advisor Mark Pauly for his invaluable advice and guidance. Mark kept a good track of my research and never hesitated to provide necessary tools. His door was always open if I ever had questions or fell behind.

I also want to extend my deepest gratitude to Peter Wonka, who helped me a lot during the later phase of my PhD journey. My visit to KAUST in late 2014 was memorable and became one of the crucial milestones during the course of my study.

Special thanks to Boris Neubert and Duygu Ceylan for being not only important scientific collaborators but also great friends. Many thanks to all LGG members for the joyful memories. Thank you Madeleine Robert for your kind support in administrative work.

I would like to take this opportunity to acknowledge Sabine Süsstrunk, Peter Wonka, Michael Wand, and Pierre Dillenbourg for being in the jury committee and providing valuable feedback in improving this thesis.

Last, but certainly far from least, I would like to thank my beloved family for their endless love, encouragement and support. Thank you Ngoc Ha for helping me to maintain a healthy work-life balance. I also thank my friends for making my PhD a wonderful experience.

My research has been supported by the funding from the European Research Council under the European Union's Seventh Framework Programme (FP 2007-2013) ERC Grant Agreement 257453, ERC Starting Grant COSYM.

Lausanne, 22 February 2016

Abstract

In recent years, the use of 3D digital content becomes widespread in various industrial and scientific domains. However, content creation still remains a costly task as extensive manual work is often required. As such, one of the core research topics in computer graphics is to accelerate the content creation process.

In this dissertation, we investigate the benefit of utilizing *shape structure* in creating different categories of digital content. To speed up the design process of 3D geometries, instead of dealing with individual shape separately, we propose to process *structured variations*, which are different models sharing certain key structural information. A geometry processing framework of structured variations consists of the *analysis* of structure from a collection of model variations, and the *synthesis* of novel shapes. We propose algorithms for three common types of digital content in computer graphics, which include facade textures, procedural modeling output, and 3D reconstruction point clouds from multiview stereo or scanning.

There is a high demand in high quality and customized facade textures, especially in urban design, 3D cities or games. We introduce a framework to create structured variations of building facades via structure-aware editing. Our framework deals with irregular facade layouts, which are common in practice.

To automatically create a large number of structured variations, a suitable technique is procedural modeling, which generates models by means of computer programs or procedures. However, the connection between the procedures and generated models is not explicit and it is usually difficult to modify the underlying procedures to generate a set of models customized to certain design intent. We present a framework which allows a user to interactively manipulate the set of generated models.

Finally, a cost-effective method to generate digital content is to digitalize the real world.

Nonetheless, the reconstructed point clouds are often noisy and contain missing data. We analyze point clouds of buildings to detect structural relations amongst building elements by means of template fitting. Once detected, these information can be used to improve the reconstruction output or to synthesize novel models via structural-aware editing.

Key words: shape synthesis, shape analysis, facade texture, structure-aware editing, procedural modeling, 3D reconstruction, template fitting

Résumé

Au cours des dernières années, l'utilisation de contenu numérique 3D c'est généralisé dans divers domaines industriels et scientifiques. Cependant, la création de contenu demeure une tâche coûteuse due au vaste travail manuel souvent nécessaire. En tant que tel, l'un des sujets de recherche de base en infographie est d'accélérer le processus de création de contenu.

Dans cette thèse, nous étudions l'avantage d'utiliser la structure des formes dans la création de différentes catégories de contenu numérique. Pour accélérer le processus de conception de géométries 3D, au lieu de traiter séparément chaque forme, nous proposons de traiter les variations structurées, qui sont différents modèles partageant certaines informations structurelles clés. Un framework de traitement géométrique des variations structurées se compose d'analyse de structure à partir d'une collection de modèles de variations et la synthèse de nouvelles variations structurées. Nous proposons des algorithmes pour trois types de contenu numérique en infographie, qui comprennent les textures de façades, la modélisation procédurale et la reconstruction 3D de nuages de points à partir de stéréo multiview ou de numérisation.

Il y a une forte demande de textures de façade personnalisées de haute qualité, en particulier pour la conception urbaine, les villes ou les jeux 3D. Nous introduisons un framework pour créer des variations structurées de façades de bâtiments via des modifications prenant en compte la structure. Notre framework traite des dispositions de façade irrégulière, qui sont courantes en pratique.

Pour créer automatiquement un grand nombre de variations structurées, une technique adaptée est la modélisation procédurale, qui génère des modèles à l'aide de programmes ou de procédures informatiques. Cependant, le lien entre les procédures et les modèles générés n'est pas explicite et généralement il est difficile de modifier les procédures sous-jacents afin de générer un ensemble de modèles personnalisés à l'intention d'un

design spécifique. Nous présentons un framework qui permet à un utilisateur de manipuler de manière interactive l'ensemble des modèles générés.

Finalement, une méthode rentable pour générer du contenu numérique est de numériser le monde réel. Néanmoins, les nuages de points reconstruits sont souvent bruités et contiennent des données manquantes. Nous analysons les nuages de points de bâtiments pour détecter les relations structurelles entre les éléments de construction au moyen d'ajustement de modèle. Une fois détectées, ces informations peuvent être utilisées pour améliorer le résultat de la reconstruction ou de synthétiser de nouveaux modèles par modification prenant en compte la structure.

Mots clefs : synthèse de forme, analyse de forme, texture de façades, édition prenant en compte la structure, modélisation procédurale, reconstruction 3D, ajustement de modèle

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Content creation is an important research topic in computer graphics and 3D geometry processing. High quality 3D shapes are needed in various industrial and scientific domains such as games, movies, material science, bio-medicine, architecture, civil and mechanical engineering. Recent advances in 3D acquisition and modeling techniques have been significantly reducing the modeling workload. Nonetheless, the modeling task is still notoriously non-trivial as it requires creativity, artistic talent and also technical skills in using the softwares. In addition extensive manual processing is usually needed and this leads to severe bottlenecks in content creation.

Recently, there is an increasing interest in creating collections of similar models such as collections of furniture, airplanes, cars or buildings with the same architectural style. These models can be used in modeling projects at different scales. For example, one needs a large number of buildings to model a 3D city. At a small scale, instead of crafting a model from scratch, the user can explore an existing shape collection to find a model that best matches his or her objectives. To create a large collection of models, manual modeling does not scale well. In this context, automatic and semi-automatic modeling methods appear to be the more suitable approaches.

The fundamental goal of this thesis work is to develop computational tools to deal with the complexity in content creation. We motivate our work by three common practices in the context of creating high quality buildings for a realistic 3D city. As this task requires a large number of models, to create each shape individually requires massive

effort and is generally impossible in practice. Nonetheless, one can observe common properties amongst different buildings such as similar facade layouts, facade elements or architectural styles. As such, in the first practical approach, instead of modeling each building from scratch, designers can follow the *modeling-by-example* paradigm and create new buildings by modifying the existing ones while preserving certain characteristic features. The second suitable approach is *procedural modeling* which aims to automatically generate a huge shape collection by means of computer programs or procedures. In this modeling scheme, generated models conform to certain global shape properties implicitly encoded in the associated computer programs. Finally, one can also reconstruct the real world, for example, by using image-based modeling or scanning techniques. However, the output of this approach is often noisy and contains missing data. Post-processing of 3D reconstruction output, therefore, is usually needed to improve the modeling quality. For instance, one can consolidate data from other similar buildings to fill up the missing parts in one building. In all these practices, buildings are not treated individually but considered as variations from the same model class. By processing those variations together, information amongst them can be reused which accelerates the design process.

A core methodological component in this thesis is to investigate the notion of *structure* in 3D geometry. Structure is the high-level information in 3D shapes, which involves both the arrangement and the relation between shape parts. Typically, models belonging to the same class exhibit common structural information. For example, a set of chairs might contain a large amount of geometric and topological variations, but the chair parts are always arranged and connected in a certain way to serve their functionality. We define *structured variations* in 3D geometries as a set of 3D models sharing some characteristic structural information while still possessing a significant amount of geometric variety.

Most of the existing geometry processing frameworks often consider structure as geometric symmetries and regularities. Geometric symmetries are results of geometric transformations such as translation, rotation, scaling or their combinations. Mathematical formalism based on the theory of transformation groups has been developed to process such structural information [66]. Despite the mathematical elegance, geometric symmetries based on group theory are limited and do not cover a vast variety of interesting structural information in practice (see Figure 1.1). A key contribution of this dissertation is to investigate *generalized structures* for example irregular facade layouts or the partial matching of parameters of pointed arches in a Gothic cathedral. We also investigate *implicit structures* which are encoded directly in the design process. Specifically, in

Figure 1.1 – Symmetry, irregular structure and implicit structure. (a) A translational pattern of identical windows (orange). This geometric symmetry can be effectively captured by the theory of transformation groups. (b) Irregular structure of windows at the Umeå School of Architecture. Although the window locations seem to be random, they are still arranged into rows and columns. We encode the irregular structures of facade elements by using facade grids. (c) Airplanes generated by procedural modeling. There exist a global implicit structure as the arrangement of wings, bodies and tails.

the rule-based procedural modeling paradigm, designers can implicitly enforce global structures via the grammars that they provide. The grammars are then passed into procedural generation engines, which, in their turn, generate shapes consistent to the intended structures. As the connection between grammars and the resulting structures is often non-obvious, one interesting question, with many practical applications, is how to customize the global structures in generated procedural models without explicitly editing the grammar rules.

We consider a *structured variation processing* framework as a means to accelerate content creation. There are different components in this processing pipeline. For each type of input data, one first need to identify the relevant types of structure and build a suitable model to represent these structural relations. The *analysis* step takes as input a given set of model variations and detect dominant structures amongst them. The *synthesis* step, on the other hand, takes core model structures as input and generates novel geometries while preserving these core structures.

Algorithms to process structured variations depends on the types of structures, which in turn relates to input data. In this thesis, we explicitly look at three types of input data for this processing pipeline, in specific, 1) building facade textures, 2) procedural modeling production, and 3) multiview stereo and scanning point clouds. These are three important types of data in computer graphics which play an important role especially in the context of urban reconstruction. We next look at the importance and challenges associated with these data in details.

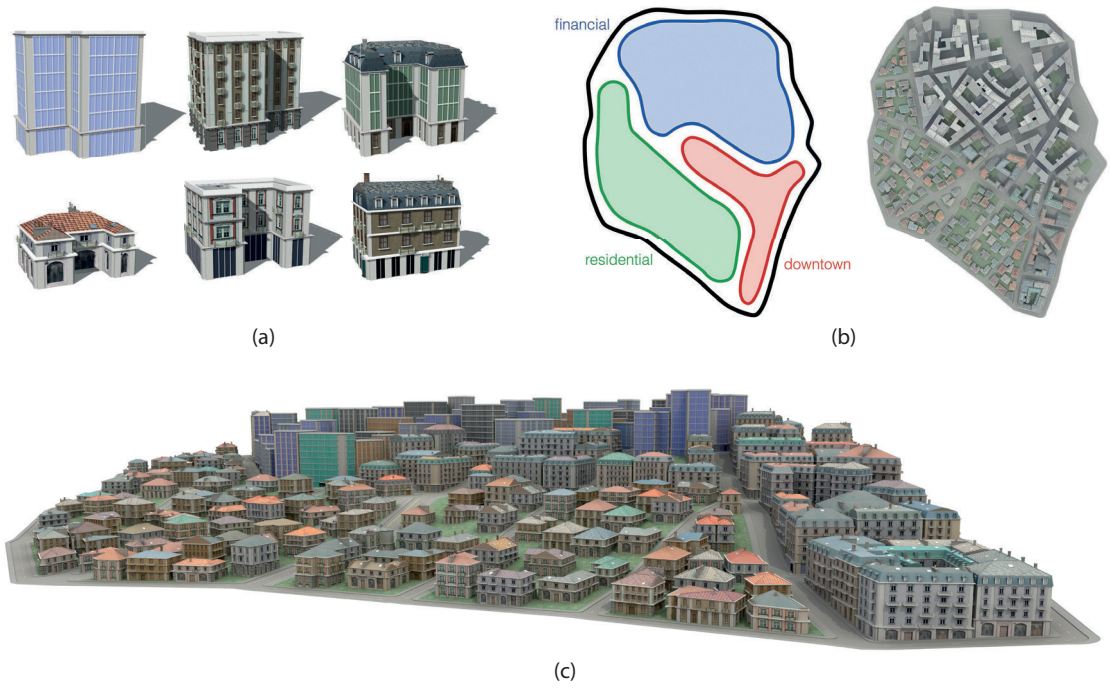


Figure 1.3 – Using a probabilistic grammar capable of generating a variety of buildings of different styles (a), by manipulating the generation output, we can design a city with 3 separate zones (financial, residential and downtown) where the house distributions are different. (b) Top view of the city. (c) Front view of the city.

1.1.1 Building Facade Textures

We first investigate the structured variations of facade textures. When creating content in digital world, there is a huge demand of high quality and customized facade textures. Once generated, these textures can be used in different applications, for example urban design, 3D cities and games. As modeling customized facades from scratch is tedious and time consuming, our objective is to make use of processing techniques in structured variations to facilitate this task. One critical challenge is the ambiguity in structured variations as there can be numerous options that satisfy user's intent (see Figure 1.2).

Figure 1.2 – Ambiguity in structure-aware editing. An exponentially large number of structured variations of the input facade (shown in red) can satisfy simple resizing operation.

1.1.2 Procedural Modeling Production

We next consider the structured variations from procedural modeling. While procedural modeling is great tool to automatically generate a large amount of structured variations, it is difficult to explicit control the generation output, in specific, the space of procedural models and the likelihood of a model being generated. A simple and interactive tool to manipulate the generation output will benefit not only expert users but also casual users who prefer to reuse existing grammars without directly editing them. In Figure 1.3 we illustrate one practical use case of the system.

1.1.3 3D Reconstruction Data

Finally, we analyze patterns of structural information in raw 3D point clouds acquired by multi-view stereo or scanning. We focus on ornate historic buildings which often contain similar elements derived from a common geometric base. For example, in Gothic churches, one can observe windows with similar arches but varying heights (see Figure 1.4). To detect this type of information, it requires a generalized concept of structures. In addition, the noise and missing data in point clouds make this analysis even more challenging.

Figure 1.4 – Structured variations from a common base geometry (pointed arch).

1.2 Contributions

In the following we summarize the main contributions of this dissertation.

- We investigate the geometry processing pipeline for structured variations in three types of common data in computer graphics: building facade textures, procedural modeling output and 3D reconstruction point clouds.

- Using the notion of generalized facade grids, we present a system to encode various symmetry, alignment and hierarchy relations among the elements of a facade. Our systems allows an interactive editing process, during which the user can specify

different grids for which our system proposes new configurations. Editing progresses by selecting such grids and one of the proposed configurations.

- We process the implicit structures encoded in procedural modeling. We present a framework that allows to interactively customize the generation likelihood of procedural models to match the design objectives.

- We investigate a generalized concept of structures as the non-local coupling in deformation parameters from a common base geometry. The framework utilizes a coupled process between template matching and deformation analysis to detect such generalized structures from raw measurement data acquired by multi-view stereo or scanning.

1.3 Organization

The remainder of this dissertation is organized as follows.

Chapter 2, Related Work. This chapter provides an extensive review of existing work related to the synthesis and analysis of structured variations, together with methods to explore the space of variations.

Chapter 3, Structure-aware Facade Editing. This chapter focuses on the generation of structured variations for building facades via a structure-aware editing framework. We deal with both irregular facade structures as well as the ambiguity in editing outputs.

Chapter 4, Interactive Design of Probability Density Functions for Shape Grammars. In this chapter, we motivate the benefits of manipulating the probability density functions of shape grammars, i.e. the likelihood that a procedural model will be generated when passing input grammar to a procedural engine. The chapter also discusses the interactive framework which enables such task.

Chapter 5, Discovering Structured Variations via Template Matching. This chapter covers generalized structures which are ubiquitous in ornate historic buildings. In addition to exact replicas, there are building elements which are structured variations from a common base geometry. To detect such structures from noisy and incomplete 3D data, the chapter introduces a novel methodology which abstracts the structures via template deformations.

Chapter 6, Conclusions. We summarize the key points of this dissertation and suggest potential directions for future work.

This thesis contains material from three published papers by the author [15, 16, 11]. In specific, Chapter 3 uses material from Reference [15], Chapter 4 from [16], and Chapter 5 from [11]. Some material from each of these papers has also been incorporated into this introductory Chapter and Chapter 2.

2 Related Work

In order to process shape variations belonging to the same 3D shape class, one option is to analyze the geometries of that shape class to identify characteristics structures and then synthesize novel shapes while maintaining these identified structural relations. Such an approach belongs to the broad domain of *structure-aware shape processing* in computer graphics. In addition, the designer can also follow the *procedural modeling* paradigm which involves automatic generation of digital contents by means of an underlying procedure. The characteristics structures of that 3D shape class can be encoded implicitly and directly into the underlying procedure. As such, the work presented in this thesis can be considered as a subset of these two larger domains in computer graphics, *structure-aware shape processing* and *procedural modeling*. In this chapter, we begin with reviewing recent work within these two domains which are related to this thesis.

2.1 Structure-aware Shape Processing

Shape structure, as defined in the Oxford dictionary, is the arrangement and the relation between parts of a shape. This high-level information connects closely to the semantic information and functionality of shapes. A typical structure-aware shape processing framework consists of two phases: 1) an analysis step to identify structural information from the input data, and 2) a synthesis process which utilizes the identified information to manipulate the input shapes, or create new shapes. An example of this pipeline is the seminal work by Gal et al. [27] which utilizes a two-step analyze-and-edit approach to modify 3D objects. Due to the importance of shape structures, a variety of structure-aware shape processing algorithms have been introduced to process different types of

input data. In the scope of this thesis, we focus on algorithms for these two data types, namely 1) 3D multiview stereo and scanning point clouds, and 2) 2D building facade textures. A more detailed overview of algorithms for other data types can be found at this survey [56].

2.1.1 Structure-aware Processing in 3D Reconstruction

To reconstruct existing physical objects, image-based multiview stereo and scanning are among the most common approaches. However, the output of such approaches is often incomplete and noisy, and rarely exposes the structure of the original models. Therefore, post-processing of the raw reconstruction output is often needed to reveal high-level structural information.

Template-based reconstruction. Physical models, especially man-made models and buildings are often constructed from similar components due to economic and style considerations. Thus, the use of templates appears as an obvious choice to analyze and improve the 3D reconstruction output. In the context of urban reconstruction, Dick et al. [19] propose a Bayesian model fitting method based on a "Lego kit" set of parametric building blocks to reconstruct 3D architectural scenes from a sequence of images. Priors on the parameters of the building blocks are predefined based on architectural rules and building styles. Schindler et al. [73] introduce another model-based approach to reconstruct CAD-like 3D building facades from images. This framework includes the selection of suitable templates from a predefined template database and the fitting of template parameters by reprojecting into the input images. Recently, Nan et al. [63] automatically select and assemble 3D templates from a prebuilt template library to reconstruct building details. For man-made models, using a set of primitive shapes such as planes and cylinders, Schnabel et al. [74] reconstruct a closed mesh from incomplete point clouds. The GlobFit system [44] proposes a primitive-based analyze and reconstruct setting, where arrangements among primitives are discovered. Lafarge et al. [39] have demonstrated a primitive-based hybrid MVS reconstruction approach for large scale models. Kurz et al. [38] propose a template deformation approach that preserves symmetry properties of templates while fitting a scanned object.

Symmetry analysis. Symmetry is ubiquitous in man-made environments and finding symmetries in geometric data has received significant attention. Transformation-space voting [54, 66] and spectral analysis [48] are among the common approaches proposed

(see the survey by Mitra et al. [55]). The method of Pauly et al. [66] is only applicable for detecting regular repetitions whereas in this thesis work we do not make any assumption about the spatial arrangement of repeated structures. The method of Mitra et al. [54] and Lipman et al. [48], on the other hand, focus on detection of exact/approximate symmetries. In contrast, our goal is to detect structured variations between input elements by discovering partial similarities across deformed templates. This is a problem that has not been addressed by any of the previous methods.

Symmetry results in redundant measurements and thus has been effectively exploited in the context of urban modeling to consolidate and improve noisy reconstructions [98, 29, 45, 93, 9]. Most of the proposed approaches, however, focus on detection of replicated elements, often arranged as regular grids. In contrast, this thesis work focuses on detection of full and partial element similarities with no assumption on their spatial arrangement.

Pattern detection. A common practice for detecting patterns is to employ the input with a set of descriptors. Leifman et al. [43] segment a given surface into pattern and non-pattern vertices using a combination of *point feature* and *curvature* histograms. Liu et al. [49] detect periodic reliefs on triangle meshes based on the auto-correlation of curvatures of the boundary points. Shechtman et al. [77] present *local self-similarity* descriptors to match images based on self-similarity of color, edges, and repetition patterns. In case of noisy and incomplete data, however, it is challenging to detect reliable descriptors. Hence, we propose an iterative approach where element similarities are abstracted by patterns in template deformations.

Co-analysis. Reconciling observations from multiple instances of data to extract reliable information is common in the literature. Learned-Miller [40] jointly aligns a set of images in a process called *congealing*. An affine transformation for each image in a stack is computed to minimize the variance for each pixel location in the stack. Faktor et al. [24] co-segment an object of interest in a given set of images by aggregating information from corresponding image patches. In the context of visual element discovery, Doersch et al. [20] use a discriminative clustering approach to detect distinctive image patches from a large set of geotagged imagery. Similarly, we present a simultaneous analysis to detect deformation patterns among a set of elements. We aggregate observations from multiple deformable templates to extract reliable similarity patterns.

2.1.2 Structure-aware Processing of Building Facades

Building facade is an important type of digital content in computer graphics, especially in the context of urban modeling and reconstruction. High quality facade textures are needed in a variety of applications, for example computer games, movies, urban planning, virtual 3D cities and digital mapping. Typical facade structures include the arrangement and the relations of facade elements, such as windows and doors. Although a facade often contains repetitive elements, e.g. a series of identical windows, the overall facade structure is in general irregular.

Facade parsing. Facade parsing is the process to detect and encode the structure in facade data. Automatic facade parsing top-down approaches often assume prior knowledge on facade models and fit the imagery cues to the models. For example, Koutsourakis et al. [34] use a Markov Random Field to fit a hierarchical tree model to rectified facade images. In a following up work, Teboul et at. [87] combine both top-down facade model fitting with bottom up image segmentation. More advanced machine learning techniques, such as reinforcement learning can be also used to improve the parsing results [88]. Semi-automatic facade parsing methods provide an alternative approach to explore the structure of facades [60]. These methods focus on decomposing a facade into a hierarchy of rectangular regions with vertical and horizontal splitting lines [59]. To effectively handle irregular facade configurations, methods that decompose the input into a set of 1D sequence of elements [47] or facade layers [97] have been proposed. Lin et al. [47] present a retargeting framework that changes the model topology by duplicating or removing the extracted 1D sequences. Recently, Zhang et al. [97] introduce a method to decompose a facade into different layers by maximizing the symmetry of the resulting substructures. This decomposition then can be used for editing operations. Finally, Lefebvre et al. [42] present a fully automatic method for extracting horizontal and vertical strips from architectural textures based on self-similarities. A number of such strips are reassembled to synthesize a new texture. However, this method does not explicitly explore any structural and hierarchical relations between the facade elements that may be desired to preserve.

2.2 Procedural Modeling

Procedural modeling is a long-standing active research topic with a wide range of applications, for example, creating textures, modeling terrain, vegetations, buildings or

city layouts. Different from traditional modeling techniques, the procedural modeling paradigm does not store the output explicitly. Instead, the digital content is encoded by compact *procedures* and only generated on demand. The underlying procedures implicitly enforce global structures over generated shapes. In addition, by varying the associated parameters, different geometries, for example a large number of buildings, can be generated. In that sense, procedural modeling can be considered as an effective tool for creating *structured variations*.

While there exist a variety of procedural methods, in this thesis, we focus on rule-based procedural modeling, in particular shape grammars and L-systems. For more details about other methods, we refer the audience to a recent survey [80].

Shape grammar. The original shape grammars were invented by George Stiny [82]. These shape grammars are defined in an analogy to phrasal grammars of linguistics [13]. A shape grammar is defined over an alphabet of shapes and generates complex shapes, in the same fashion as a phrasal grammar generates strings of symbols from an alphabet of symbols. Starting from the original *start shape*, a shape grammar uses shape rules to transform all geometrically matched parts of the existing shape. This generation strategy has been used to successfully generated a variety of designs, for example the Palladian villa plans [84], the bungalows of Buffalo [22], or the Malagueria grammar [23]. The original shape grammars, however, are too complicated for most modeling tasks. Therefore, most work in computer graphics is based on a simplified version of shape grammars, *set grammars* [83]. Different from shape grammars, a set grammar works on a combination of labelled shapes and in each step transform a labelled shape without looking for geometric matching. In specific, a set grammar G consists of two sets of shapes T (terminal shapes) and N (non-terminal shapes), and a set of shape rules R. The shapes in T and N are n-dimensional geometries. Each shape rule in R transforms a shape in T to one or multiple shapes in $T \cup N$. Starting with a non-terminal shape, often named *axiom*, the shapes are recursively replaced by applying rules in R until all the shapes are terminal shapes. In comparison to the original shape grammars, set grammars are easier for computer implementation [28] and recently several grammars have been proposed especially for modeling streets and buildings, e.g. [65, 91, 57].

Parametric and stochastic shape grammar. To extend the modeling capability of a shape grammar, one can associate each terminal and non-terminal shape with a set of descriptive parameters. Examples of these parameters are the width and height of building windows or the relative position of airplane wings. In addition, multiple shape

rules can be associated with a shape to transform it in different ways. By randomly choosing the parameters and the rules, shape variations can be generated.

CGA grammar. In the context of urban modeling, one of the most developed grammars is the *Computer Generated Architecture* (CGA) grammar introduced by Müller et al. [57]. This grammar is specially designed for the procedural generation of buildings, and has been integrated into a commercial software Esri CityEngine [1]. The CGA grammar is a shape grammar which is extended from the parametric context-free *split grammar* previously proposed by Wonka et al. [91]. A shape (both terminal and non-terminal) in this grammar is associated with an oriented bounding box called scope and optionally some geometry located by this scope. Shape rules sequentially transforms existing shapes to refine the model. CGA grammar also includes specialized operations to generate buildings for example roof operation or occlusion handling. Although CGA grammars have been successful in generating visually plausible buildings with various styles, there are still some limitations associated. For example, it does not coordinate the rule applications across multiple shapes e.g. to generate only one door in the ground floor. CGA grammar does not handle rounded layout either. Some improved grammars have been proposed, notably the *Generalized Grammar* (G^2) [37], and *CGA++* [75].

L-systems. L-systems are very similar to grammars but they use a parallel replacement strategy instead of a sequential one. They have been successfully used for modeling plants [68, 69] and have been extended to query and interact with their environment during derivation to tackle more challenging plant modeling problems [70, 61]. Similar to shape grammars, parametric and stochastic L-systems are also developed to enhance the generation capability.

Inverse procedural modeling. Currently, high-quality grammars are predominately written by hand. Inverse procedural modeling focuses on creating procedures from example shapes which are then used to synthesize similar shapes. One approach is to analyze the patterns from a set of training models and generate deterministic rules. For example, parametric context-free L-systems for 2D vector images are automatically generated by detecting similar elements such as curves and poly-lines and assigning corresponding symbols to establish an L-system alphabet [81]. Partial symmetry in 3D geometry can be also extracted to build a shape grammar, which is then used to create new model variations [6]. Another approach to induce shape grammars is based on Bayesian inference using a minimum description length prior on the grammar structure [86, 52]. The grammar

[1] http://www.esri.com/software/cityengine

structure can be optimized using randomized algorithms (such as Markov chain Monte Carlo) with local moves including both *rule splitting* and *rule merging* operations. In general, there is a trade-off between grammar complexity and expressiveness. Algorithms that try to learn grammar structure or grammar parameters tend to build on very simple, typically context-free grammars or only learn rule parameters [58, 79, 94].

Remarks. While procedural modeling methods, especially rule-based techniques, are capable of compactly encoding global structural relations over the generated geometries, these relations are often implicit and not obvious to designers. In addition, while variations can be created by by parametric stochastic grammars, to obtain a set of shape variations for particular design intent, it often requires tedious manual work in tuning grammar parameters or editing the rules directly. One of the challenges in procedural modeling is to provide intuitive control for designers.

2.3 Exploring Structured Variations

Another key task in processing structured variations is to explore the space of these variations. This task falls into the domain of exploratory modeling in computer graphics. An early inspiration for these efforts is the concept of design galleries introduced by Marks et al. [51].

An important category of exploratory modeling efforts focus on exploring a pre-defined, discrete design space such as a collection of websites [41] or 3D shapes [33]. Several papers have proposed to use high-level feature attributes and utilized crowdsourcing tools to learn the relevance of such attributes [64, 12]. Averkiou et al. [1] compute a hierarchical embedding of a large shape collection. In addition to exploring the given shape collection, they also compute the most dominant variation modes in this embedding as basis vectors. They utilize these basis vectors to define an inverse mapping from the embedding to the shape space and to generate new shapes. Our algorithm, on the other hand, operates on large design spaces defined by production grammars or other generative processes.

One approach to explore large design spaces is to provide the users with a discrete set of samples in an interactive framework. Samples can be generated by utilizing probabilistic models [53] or evolutionary algorithms [95]. In another thread, researchers have proposed to locally explore the design space in the neighborhood of an optimized sample with respect to an energy function [89, 96, 18]. Bao et al. [4] have extended this idea to

enable global exploration by extracting good pathways between different local spaces. In the context of procedural modeling, Lienhard et al. [46] propose a strategy to sample a procedural space to generate representative thumbnail images.

Several researchers have adopted a parametric model to explore large design spaces assuming a direct mapping between the changes in the parameters and the output design. For example, Kerr and Pellacini [32] have proposed to use sliders for the parameters of the design space to help the users select different materials. Koyama et al. [36] fit a goodness function by defining a goodness value on a set of discrete samples based on crowdsource data and interpolating these values in the corresponding parameter space using radial basis functions. Kovar and Gleicher [35] aim to construct the *legal* space of PostScript drawings by requiring user feedback on a set of initial samples and generate new samples by interpolation in the corresponding parameter space. Talton et al. [85] explore parametric design spaces of trees and human shapes where they focus on avoiding invalid shapes by defining a density function based on manually selected valid models. Shapira et al. [76] present an exploratory interface for recoloring images by parameterizing the design space using Gaussian Mixture Models. Brochu et al. [8] present a Bayesian optimization approach to explore parametric animation spaces.

Remarks. This thesis work is inspired by these research efforts focusing on exploration of large design spaces, in particular exploring the structured variations in building facades and in the procedural production. One core challenge is ambiguity in structured variations as there often exists multiple outputs that might match the design intent. In addition, in procedural modeling, the procedural production is neither pre-generated nor can be mapped into a parameter space. Thus, existing techniques cannot be applied to explore the space of procedural models. We address these challenges in our work.

3 SAFE: Structure-aware Facade Editing

Figure 3.1 – Structure-aware facade editing. Using the notion of generalized grids, our system encodes various symmetry, alignment, and hierarchy relations among the elements of a facade. During incremental editing, the user can specify different grids (shown as box abstractions) for which our system proposes new configurations. Editing progresses by selecting such grids and one of the proposed configurations (shown in red).

In this chapter, we introduce a framework to synthesize structured variations for building facades by editing input facade textures. Many man-made objects, in particular building facades, exhibit dominant structural relations such as symmetry and regularity. When editing these shapes, a common objective is to preserve these relations. However, often there are numerous plausible editing results that all preserve the desired structural relations of the input, creating ambiguity. We propose an interactive facade editing framework that explores this structural ambiguity. We first analyze the input in a semi-automatic manner to detect different groupings of the facade elements and the relations among them. We then provide an incremental editing process where a set of variations that preserve the detected relations in a particular grouping are generated at each step. Starting from one input example, our system can quickly generate various facade configurations.

3.1 Foreword

One of the long-standing problems in computer graphics is to provide artistic control for content creation. Modeling of shapes is not trivial because it requires both artistic talent and technical skill. The design process is time-consuming and error-prone as extensive manual processing is often needed to obtain high-quality models. To address these challenges, recent research efforts focus on the *modeling-by-example* paradigm, where the goal is to modify an existing model to create new shapes while preserving certain features of the original shape. Such a paradigm is particularly useful for modeling urban spaces, since many applications (e.g. mapping and navigation, urban design, content creation for entertainment) can benefit from a fast and easy design process.

Building facades often exhibit dominant structural relations such as symmetry and regularity. When producing new shapes by editing a given example, these relations should typically be preserved. However, this is a highly ambiguous process as there are often multiple ways to maintain the structural relations that all result in plausible output shapes. Amongst these shapes, there is no definitely correct output and the desired solution depends on the intent of the user. Therefore, instead of committing to a particular output based on certain heuristics, it is vital to be able to efficiently navigate through alternative solutions.

In this chapter, we present an interactive framework for structure-preserving editing of 2D building facades that enables exploration of the structural ambiguity during the editing process. We assume as input an ortho-rectified facade image that has been hierarchically segmented with vertical and horizontal splitting lines into rectangular sub-regions. Certain sub-regions such as the windows and doors are semantic facade elements and we preserve the arrangements of these elements during editing.

It is often desirable to edit a group of related elements together. For example, identical windows arranged in a regular grid are typically expected to behave similarly. A row of windows and the door separating them might act as a grid of nonidentical elements if grouped together, making the insertion or deletion of either of the element types possible. Often, there are multiple ways to group a set of elements and the particular grouping of interest depends on the user intent. Thus, we provide a semi-automatic framework to group the facade elements. Given a particular grouping, we support editing operations such as insertion or resizing of elements, while propagating the edits to hierarchical sub-elements.

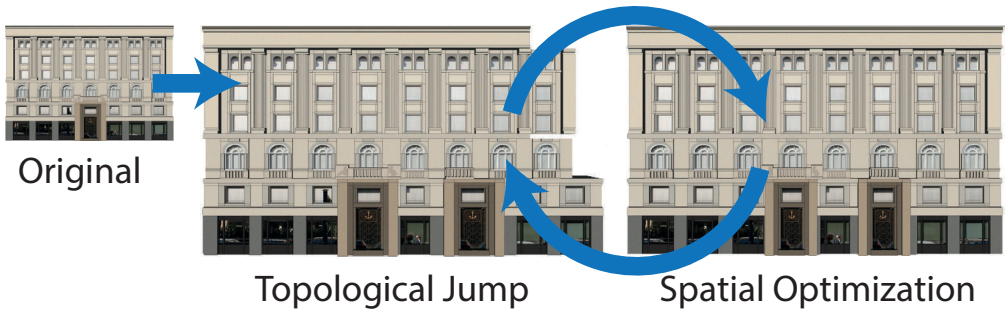


Figure 3.2 – The proposed editing pipeline consist of two steps: The topological jump step explores structural variations of the input facade by changing the number of facade elements. Then, in the second step spatial configurations of the facade elements are optimized to generate a plausible facade.

3.2 Contributions

Our main contribution is a novel incremental editing process that exposes shape ambiguities by prompting the user with a set of alternative output shapes at each editing step. A central feature of our approach is the ability to encode the structure of a facade as a general group of elements that can be nested in a hierarchy. This avoids limitations of most existing systems that restrict editing operations to regular grids only. We evaluate our framework on building facades of varying complexity and demonstrate that a large variety of plausible output shapes can easily be created from a single input example.

3.3 Overview

One of the core challenges in structure-aware editing is ambiguity: there are often multiple consistent ways to maintain a set of structural relations. Simple operations, such as resizing a facade, can quickly lead to a combinatorial explosion of possible solutions. Many existing editing methods provide a single solution based on a set of heuristics that try to anticipate the intent of the user. However, the user might initially only have a vague idea of the desired output. In such a case, the final solution can be obtained in an exploratory process by iterative refinement of intermediate results. Therefore, our aim is to give access to a large space of possible solutions, while avoiding exposure to an exponential set of variations. We achieve this goal with an incremental editing process that prompts the user with a small set of variations at a time. The output is successively refined by selecting one of these variations at each step.

We distinguish between two fundamental types of modifications to a facade: *discrete* modifications that change the number of facade elements, e.g. inserting new elements or removing existing elements, and *continuous* modifications that change the size of facade elements, e.g. resizing a window. We propose an editing framework that enables such modifications in a two-step approach. Discrete modifications of the input facade are performed in the **topological jump** step, while continuous modifications are applied to the resulting facade elements in the **spatial optimization** step to compensate for the distortion introduced by the structural changes. This separation allows a stepwise exploration of structural variations while the potential variations due to continuous changes in size are ignored and expressed as constraints in the optimization. Note that only the spatially optimized facade configurations are exposed to the user.

3.4 Representing Spatial and Structural Relations

In this section we present the data structures that capture spatial and structural relations between facade elements. Such relations are preserved in the editing operations we present. Facade parsing algorithms provide a method to decompose a facade into smaller shapes by recursive subdivision. In the following we call this spatial subdivision structure *decomposition tree*. By introducing *parent-child* relations between facade elements, this data structure captures how changes in the size of a facade element induce changes at its parent. Besides the spatial relations, facade elements exhibit structural relations, e.g. the number of windows in one floor matches the number of windows in the second floor, which might or might not be of the same kind. We introduce an additional data structure (*facade grids*) to capture this information.

Spatial Decomposition: Decomposition Tree. Many facade parsing [59, 78] and element classification [72, 88] algorithms result in a spatial decomposition of the input data. This decomposition is usually represented as a tree with alternating splitting directions. A node in the tree represents a *facade shape* associated with a rectangular area. The root node (representing the whole facade) is recursively subdivided into smaller rectangular shapes by splitting along the x or y directions. We store the direction of the subdivision and its relative position with respect to the size of the node (*split lines*).

The resulting spatial decomposition exhibits properties that we exploit during the optimization step to compute valid spatial configurations of new facade variations (see

Sec. 3.6). The *total decomposition property* allows us to express the size of a node in terms of the size of its children. Specifically, if a node has x (y) splits, the width (height) of this node is equal to the sum of the widths (heights) of its children, where as the node and its children have identical heights (widths).

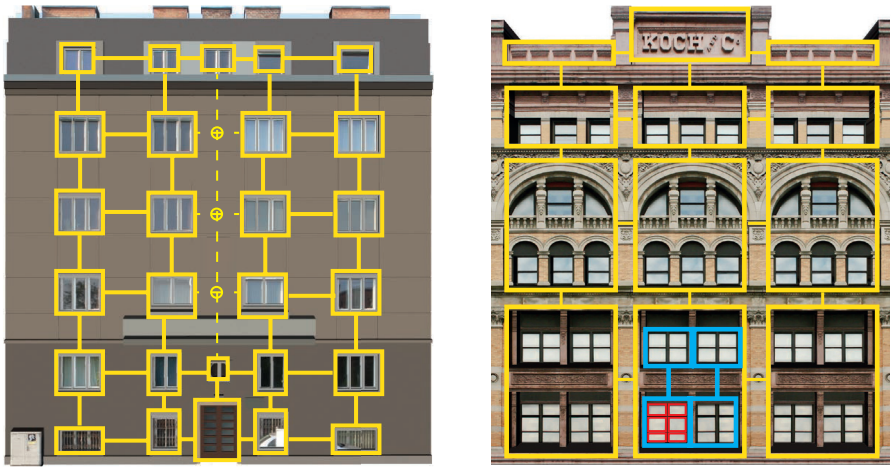
Several automatic or semi-automatic methods exist to define and detect splitting lines. We refer the reader to the survey on urban reconstruction for a thorough review of this topic [60]. Another source for such spatial decompositions are facade configurations resulting from shape grammars that can be naturally transformed into decomposition trees based on the grammar parsing tree of the grammar (such as [58]). We provide examples using facade decompositions resulting from both methods.

Structural Relations: General Grids. We present a data structure to capture the structure of the input facade and influence the resulting variations. Earlier attempts consider structural information in terms of symmetries, repetitions, or regularity of a model using one- or two-dimensional regular lattices (cf. [7, 66]). Although these approaches can be applied to a wide range of models, they fail to encode potential links between non-symmetric elements (e.g. non-regular element spacing) and cannot encode hierarchical relations (e.g. a regular element itself consists of a regular configuration of subelements).

Horizontal and vertical alignments are typically most important to define the structure of building facades. We encode these alignments by a grid-like data structure. Grids provide an intuitive way to specify structures in facades as well as the constraints between facade elements. Elements that are part of the same grid are meant to behave similarly under structural changes. Such relations are used in the following sections to constrain the spatial optimization.

We employ a generic definition for these grids where grid elements are nodes of the facade decomposition tree. These elements do not need to be similar, can be unevenly spaced (Figure 3.3), and be part of more than one grid. A *facade grid* is defined as a group of non-overlapping facade elements, which are arranged into columns and rows. The user either manually selects elements that should be combined to a grid, which allows arbitrary elements in a grid, or nodes similar to a selected element are identified based on automatic symmetry detection methods [9]. This generic definition of facade grids enables the grouping of different types of elements, and hence makes it possible to handle

facades without dominant repetitions (see Fig. 3.13).



(a) (b)

Figure 3.3 – (a) A general grid with different types of elements is shown. Dashed lines connect phantom elements (circles) in the middle column. (b) The grid in red is a subgrid of the blue grid, which in turn is a subgrid of the yellow grid.

Given a group of facade elements, we assign each of them a unique coordinate consisting of a row and a column index. For column assignment, starting with the left-most element, we consider the next element to belong to the same column as long as we observe a vertical overlap between the elements. These elements form the first column. We repeat this process for the remaining elements to obtain additonal columns. The assignment of row indices is similar, starting with the top element. Our scheme does not require all rows or columns to have the same number of elements. In the case of missing elements we add a *phantom* element as a place holder without assigned geometry to obtain a rectangular grid configuration (see Figure 3.3a).

Structures can be observed at different levels within a hierarchy, i.e. one structure can be contained within another structure (see Figure 3.3b). We translate this hierarchical relation into a *hierarchy of facade grids*. These hierarchical configurations are automatically assigned if all elements of a grid are included in the subtree (of the decomposition tree) of an element of another grid. The former grid is then called the subgrid of the latter one.

3.5 Discrete Modification – Topological Jump

The objective of our incremental editing framework is to generate plausible facade variations that preserve desired structural relations between facade elements. We enable the user to select a set of facade grids at each editing step for which the algorithm will suggest new configurations. Such selected grids are called *active grids*. When a grid is selected, our framework enables automatic selection of other grids with identical element types and counts. By selecting different active grids, the user specifies different structural relations to be preserved at each editing step and explores the resulting variations. After choosing one of the proposed variations, the user proceeds by changing the active grids or analysing further extensions of the current active grids. In this section, we describe how new facade configurations are generated by changing the number of elements in an active grid by utilizing both structural and spatial information. Note that the size of the elements in the new facade configuration are determined in the *spatial optimization* step (see Section 3.6).

Once a grid is selected to be active, our method first examines the content of the grid to determine its possible variations. Specifically, if the editing operation increases the width (height) of the facade, the unique columns (rows) as potential insertion candidates are identified. We call such unique columns (rows) the *source* columns (rows). Insertion of any of the source columns (rows) in each possible location of the active grid results in a potential variation presented to the user. In the following subsections, we describe how discrete modifications to an active grid are performed. For convenience, we only describe the case where an editing operation changes the width of a facade. Changes in the height of the facade are handled in a similar fashion.

Structure-aware Insertion Operation. Insertion of a new grid column is performed by insertion of each element in the column in a row-wise manner. Therefore, we first describe how a grid element S which we call the *source* element is inserted between two *anchor* elements A_l and A_r. Often, grid elements such as windows are most prominent elements adjoining less important non-grid elements such as walls. Thus, insertion of a new grid element requires the duplication of the surrounding content of the anchors to ensure that the source is embedded in a region similar to the neighborhood of the anchors before the insertion (see Figure 3.4). Preserving such neighborhood relations requires direct access to the neighborhood information of each grid element. While the facade decomposition tree provides hierarchical decomposition links between the

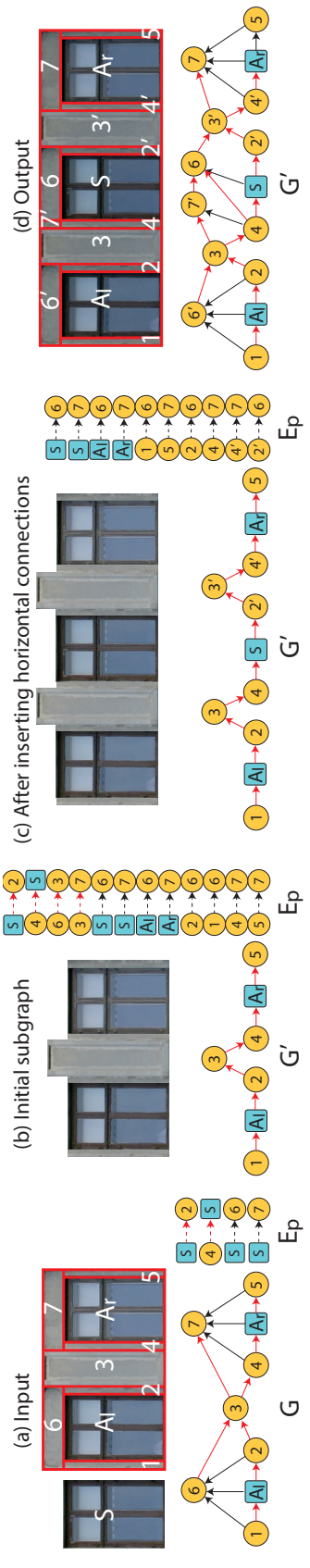


Figure 3.4 – The source S is inserted between the anchors A_l and A_r. The neighborhood graph G' and the contents of the pending edge queue E_p is shown at each step of the insertion process. Vertical and horizontal edges are in black and red respectively.

facade elements, it does not provide direct access to the neighborhood information as neighboring elements may be part of different subtrees depending on the order of the subdivision operations. Instead, we encode the neighboring relations between facade elements in a graph structure called the *neighborhood graph*.

A neighborhood graph $G = (N, V, H)$ is a directed graph composed of a set of nodes N where each node corresponds to a facade leaf shape in the facade decomposition tree. A vertical edge $e_v = (n_i, n_j) \in V$ (horizontal edge $e_h = (n_i, n_j) \in H$) directed from n_i to n_j connects these two nodes if the corresponding facade shapes share a vertical (horizontal) boundary and n_i is below (to the left of) n_j. The neighborhood graph can be considered as a dual structure of the facade decomposition tree which provides direct access to relative positions of the graph nodes. It is straightforward to build this graph given a decomposition tree. Conversion of a neighborhood graph G to a facade decomposition tree, on the other hand, is performed in a recursive manner. At each step of the conversion, the longest sequence of vertically (or horizontally) connected nodes $C = \{n_0, ..., n_k\}$ is extracted such that all nodes $\{n_0, ..., n_{k-1}\}$ have only one outgoing edge and all nodes $\{n_1, ..., n_k\}$ have only one incoming edge of the same type. Such a sequence of nodes, called *chain*, are collapsed to a single node and a new parent shape is added to the decomposition tree to represent the collapsed node. A chain is equivalent to a set of sibling nodes in a facade decomposition tree. Thus, if G represents a valid facade decomposition tree, it is ensured that a chain can be detected at each step. The conversion process terminates when the whole graph is collapsed to a single node which represents the root of the corresponding decomposition tree.

When inserting a source element S between the anchor elements A_l and A_r, we duplicate additional facade shapes and determine their spatial arrangement in the new facade configuration by utilizing the neighborhood graph. Specifically, we build a neighborhood graph G corresponding to the subtree of the facade decomposition tree rooted at the common parent of A_l and A_r, since this subtree contains all relevant elements. Insertion of S is then carried out by constructing a new neighborhood graph G' from G which contains S. S is embedded in G' in such a way that its neighborhood is similar to those of A_l and A_r in G. One naive approach to obtain G' is to connect the source S to all neighbors of A_l and A_r. However, this often leads to an invalid graph, i.e. a graph which does not represent a valid facade decomposition. For example, an element might end up as both right and left neighbors of another element (see Figure 3.5). This observation supports the intuition that S can be embedded in a neighborhood similar

to the neighborhood of both anchors only by duplicating some nodes in G. Therefore, we propose an incremental solution that takes a valid neighborhood graph as input and adds new edges one at a time while ensuring that the graph remains valid at each step. Necessary nodes are automatically duplicated during this process. Once G' is constructed, we convert it back to a decomposition subtree to replace the original subtree. Next, we describe the details of the incremental edge insertion process.

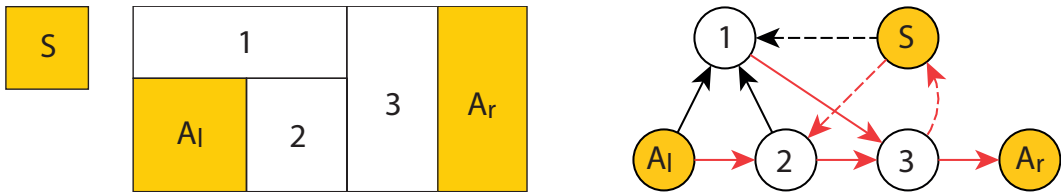


Figure 3.5 – Connecting the source S to all the neighbors of the anchors A_l and A_r results in a conflict: the loop $(S, 2, 3)$ suggests that S is both to the left and right of 3. (Vertical and horizontal edges are shown in black and red respectively.)

Incremental Edge Insertion. Given an initial neighborhood graph G including the anchors A_l and A_r, our goal is to insert the source S between the two anchors. To achieve this goal, we incrementally construct a new neighborhood graph G' by utilizing a *pending edge queue*, E_p, consisting of the edges that need to be added to G'.

First, the edges in G that involve either A_l or A_r are added to E_p while replacing the respective anchor node with S (see Figure 3.4 a). Insertion of these edges in the subsequent stages influences the neighborhood of S to be similar to the neighborhood of the anchors. We then initialize G' as the subgraph extracted from G consisting of the longest sequence of horizontally connected nodes including A_l and A_r. Any edge of G that is not included in this subgraph is added to E_p (see Figure 3.4 b). Once E_p is initialized, edges in E_p are inserted to G' incrementally. Inserting an edge requires an update of the neighborhood information to ensure a valid facade configuration at any point during the process. Please note that a new edge in E_p becomes available for insertion when at least one of the nodes it connects is present in the current graph.

Edge insertion starts with the pending horizontal edges, processing those involving S first. Next, the pending vertical edges are processed. If a node is required to be vertically connected to a sequence of horizontal nodes in G', all such vertical edges are inserted simultaneously (e.g. $e(S, 6)$ and $e(2', 6)$ in Figure 3.4 c are inserted at once). The

algorithm continues by processing the remaining pending edges in a similar manner until E_p is empty.

Once a pending edge is selected, its insertion is performed based on whether it involves one or two nodes present in the current graph G'. Assume a horizontal edge $e(n_i, n_j)$ is selected that connects a new node n_i to an existing node n_j. Intuitively, insertion of such an edge is equivalent to introducing a vertical split in the facade shape of n_j to create a tiny sub-shape and replacing this sub-shape with the facade shape of n_i. In other words, our goal is to correctly embed the new node n_i between the nodes already present in G' with coherent horizontal and vertical relations. Thus, we first extract all incoming horizontal edges of n_j and relink them to n_i (in Figure 3.6, $e(1, 8)$ and $e(2, 8)$ are relinked to $e(1, N)$ and $e(2, N)$). If such a relinked edge was present in the original graph G, we add it to E_p so that this neighborhood information is not lost. We next establish the vertical neighborhood relations of n_i by connecting it to one of the above and one of the below neighbors of n_j. Specifically, we extract a set of nodes M_{below} that are connected to n_j with vertical incoming edges and a set of nodes M_{above} that are connected to n_j with vertical outgoing edges. Since all nodes in M_{below} (or M_{above}) are connected to n_j vertically, there is a horizontal path connecting all of them. We connect n_i vertically to the leftmost node along such a path (in Figure 3.6, vertical edges $e(N, 4)$ and $e(6, N)$ are established).

The described procedure so far allows to insert an edge with a new node into G' and update the present edges to ensure a valid configuration. If at any point of this process, an edge which has been previously added to G' from E_p is required to be pushed back to E_p, this insertion is discarded and we undo any update in G' triggered by this insertion. This ensures that no edge is inserted to G' and pushed back to E_p continuously.

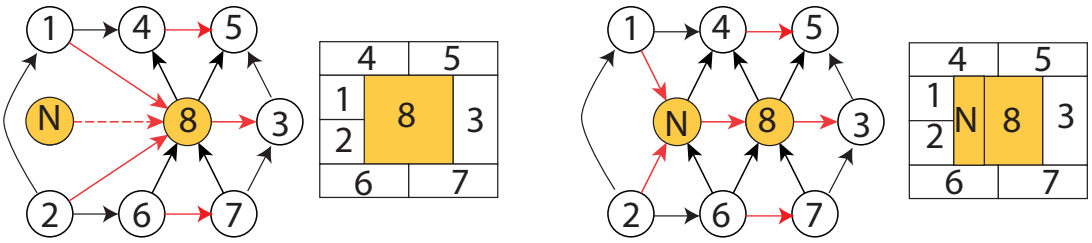


Figure 3.6 – The edge $e(N, 8)$ is inserted to the current neighborhood graph. Vertical and horizontal edges are shown in black and red respectively.

If the selected pending edge connects two existing nodes n_i and n_j in the current graph G', its insertion might result in a conflict. In order to avoid such conflicts, we duplicate one of the edge nodes n_i and insert the pending edge between the new duplicated node n_i' and n_j instead. This edge now connects a new node n_i' to the existing node n_j and can be added as described before. When a node is duplicated, all the edges of the original node are copied and added to E_p. Note that if both nodes of a pending edge have already been duplicated, we discard it to ensure that a node is duplicated at most once.

Coupled Insertion in Multiple Rows. In cases where anchor elements in different rows of a facade grid have the same common parent, performing the column insertion row-by-row will result in multiple duplications of certain facade shapes. To avoid such a scenario, we perform the element insertion in these rows in a coupled fashion. We first find the subtree rooted at the common parent of the identified rows and construct the corresponding neighborhood graph G. Intuitively, we divide G into multiple subgraphs where each subgraph consists of a single grid row. We insert source elements into corresponding subgraphs and combine the updated subgraphs. In more detail, we extract a set of subgraphs $G^s = \{G_1, ..., G_m\}$ where G_k represents the longest sequence of horizontally connected nodes in G including the anchor elements in the corresponding row r_k. We then initialize a common pending edge queue E_p with any edge that is not included in any subgraph in G^s. For each source element to be inserted, we also copy the edges from the corresponding anchors and add to E_p. We then label the edges in E_p to denote an order of insertion to the extracted subgraphs. Specifically, starting from the top row G_1, we define a node set N_k as the set of nodes in G_k and the unlabeled nodes that are above any node in G_{k+1}. Edges that connect any two nodes in N_k are labeled as l_k. At the end of this grouping, a set of edges remain unlabeled which are the vertical edges later used to connect the updated subgraphs in G^s. Next, starting from the first row, the incremental edge insertion process described before is performed by updating the corresponding subgraph of the row. The only notable difference is that when updating the graph G_k, only the edges labeled l_k are processed.

Once all rows are processed, E_p contains only the unlabeled edges. We use these edges to combine the subgraphs in G^s into a common graph G'. A pair of vertical edges are used to connect two subgraphs if they are not *crossing*. Edges $e(x_1, y_1)$ and $e(x_2, y_2)$ are defined to be crossing if x_1 is located to the left of x_2 and y_1 is to the right of y_2. Once all updated subgraphs are combined into G', it is converted to a decomposition tree to replace the original subtree.

3.6 Continuous Modification – Spatial Optimization

In the topological jump step of our framework, new facade elements are added to the spatial decomposition tree and the respective facade grids. This process violates the total decomposition property (as described in Section 3.4) and thus requires an update of the size of the resulting nodes (see Figure 3.2). The naive option to compute valid sizes for the nodes is to propagate the change due to additional elements up to the root node, effectively increasing the size to accommodate space for the new elements. Further, changes in inner nodes need to be propagated down to the children, e.g. by evenly distributing the size increment. While this re-establishes the total decomposition property of the spatial data structure, the results often violate aesthetic properties such as alignment and coherent size of similar elements (see Figure 3.7). To address this issue, we propose an optimization scheme that minimizes the deviation from the original size of the elements while respecting additional constraints relating the size and the alignment of the elements in facade grids. We formulate this optimization as a *quadratic programming* problem.

Figure 3.7 – Simple propagation of changes along the decomposition tree (a) breaks alignments between facade elements, which can be preserved by our spatial optimization (b).

Our optimization process shares some similarities with the method of Bao et al. [3] with one fundamental difference. We optimize for the size of all the facade shapes at once while Bao et al. recursively solves for a sequence of 1D layout problems. The size of the already processed facade shapes impose certain layout constraints for the subsequent stages of their algorithm. Thus, backtracking is necessary if a solution cannot be found at a certain step.

At the end of each editing step, we compute the new width and height of the facade shapes by setting up two independent optimization problems respectively. In the following, we describe how the new shape widths are computed. Computation of the shape heights is performed similarly by interchanging width and height and x and y-splits.

Quadratic Programming. When computing new sizes of facade shapes, our goal is to preserve the original sizes as close as possible. Thus, we define the quadratic objective $E(\mathbf{x}) = \|\mathbf{W}(\mathbf{x} - \mathbf{x}^*)\|^2$, where $\mathbf{x}$ and $\mathbf{x}^*$ respectively denote a vector of new and original widths of the facade leaf shapes and $\mathbf{W}$ is a diagonal weight matrix. We weight the changes in shape sizes by the inverse of their areas to allow larger shapes to deviate more. We normalize the shape areas by dividing by the total facade area and cap the weights at 10.0. We minimize $E(\mathbf{x})$ for positive $\mathbf{x}$ subject to hierarchy constraints C_h, alignment constraints C_a and symmetry constraints C_s as described next:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & E(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} > \mathbf{0}, \mathbf{C}_h\mathbf{x} = \mathbf{0}, \mathbf{C}_a\mathbf{x} = \mathbf{d}_a, \mathbf{C}_s\mathbf{x} = \mathbf{0} \end{aligned} \tag{3.1}$$

Additional constraints specified by the user such as the target size and location of a shape can be integrated into Equation 3.1 to provide additional interaction possibilities.

Hierarchy Constraints. We specify a hierarchical link between the size of a facade shape and its children. For each leaf shape, we define a binary vector α with one non-zero entry corresponding to the width, w, of the leaf in $\mathbf{x}$, such that $w = \alpha^T\mathbf{x}$. The width of an internal node is then obtained from the binary vectors α of its children. Specifically, if S has an x split, we obtain $\alpha_s = \sum_i \alpha_{c_i}$ and $w_s = \alpha_s^T\mathbf{x}$. In case of a y split, however, any two children c_i and c_j have the same width, $\alpha_{c_i}^T\mathbf{x} = \alpha_{c_j}^T\mathbf{x}$, equal to the width of S. Thus, we set $\alpha_s = \alpha_{c_0}$, and for any pair of children (c_i, c_{i+1}), we represent the constraint $(\alpha_{c_i} - \alpha_{c_{i+1}})^T\mathbf{x} = 0$ as a row of $\mathbf{C}_h$.

Shape Alignment. Facade elements that belong to the same grid need to preserve their relative positions during the editing process. Therefore, we define constraints relating the pairwise distances between neighboring grid elements. To quantify these alignment constraints, we first associate each grid element S with an anchor point A_s. In horizontal edits, for each column, we define three sets of points consisting of (i)

the centers, (ii) midpoints of the left edge, and (iii) midpoints of the right edge of the grid elements in the column. We use the point set with minimal variance in terms of x-coordinates as the anchor point set. The shape alignment constraints preserve the distance between the x-coordinates of these anchor points.

The x-coordinate, a_s, of the anchor point A_s of a shape S can be expressed in terms of the x-coordinate, l_s, of the left edge of S:

$$a_s = l_s + \epsilon w_s, \tag{3.2}$$

where $\epsilon = 0$ if A_s is the midpoint of the left edge, $\epsilon = 0.5$ if it is the shape center, and $\epsilon = 1$ if it is the midpoint of the right edge. Further, l_s can be computed as:

$$l_s = l_p + \sum_{s_i \in \mathbb{S}_S^-} w_{s_i}, \tag{3.3}$$

where l_p is the x-coordinate of the left edge of the parent of S, $\mathbb{S}_S^-$ is the set of siblings in the facade decomposition tree located to the left of S. w_{s_i} denotes the width of such sibling nodes. Setting $l_{root} = 0$, l_s is computed as a linear combination of the widths of the leaf nodes $\mathbf{x}$: $l_s = \beta_s^T \mathbf{x}$. By plugging this expression and $w_{s_i} = \alpha_{s_i}^T \mathbf{x}$ into Equation 3.3 we obtain

$$\beta_s = \beta_p + \sum_{s_i \in \mathbb{S}_S^-} \alpha_{s_i}. \tag{3.4}$$

Finally, from Equation 3.2, we have

$$a_s = \beta_s^T \mathbf{x} + \epsilon \alpha_s^T \mathbf{x}. \tag{3.5}$$

We then express the alignment constraints between any two consecutive grid element S_i and S_{i+1} in a column in terms of the x coordinates of the corresponding anchor points: $a_{s_i} - a_{s_{i+1}} = d_{i,i+1}$, where $d_{i,i+1}$ is the original horizontal distance between A_{s_i} and $A_{s_{i+1}}$. This translates into $(\beta_{s_i} + \epsilon\alpha_{s_i} - \beta_{s_{i+1}} - \epsilon\alpha_{s_{i+1}})^T \mathbf{x} = d_{i,i+1}$ and is represented as a row in the system $\mathbf{C}_a \mathbf{x} = \mathbf{d}_a$.

Shape Symmetry. Certain facade elements, either manually indicated by the user or detected by automatic symmetry detection methods, are desired to have the same size.

We formulate this symmetry constraint between two such shapes S_i and S_j using their α vectors: $(\alpha_{s_i} - \alpha_{s_j})^T \mathbf{x} = 0$.

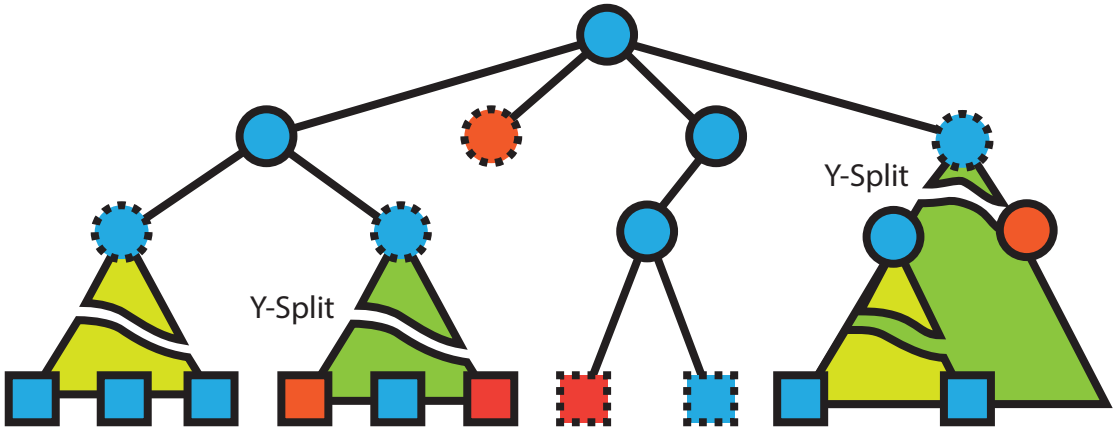


Figure 3.8 – Variable reduction process: Yellow subtrees are collapsed as they do not contain any node involved in a constraint (shown in red). Green subtrees are collapsed as the parent node is subdivided into a set of leaf nodes with y splits. The leaf nodes in the final tree are shown as dotted and are used as variables in the optimization process.

Variable Reduction. The widths of all the facade leaf shapes are considered as variables in our optimization. However, we can reduce the number of variables by analyzing the constrained nodes of the facade decomposition tree. Specifically, we collapse the subtree rooted at a node n_i if none of the remaining nodes in the subtree is involved in a constraint (see Figure 3.8, yellow subtrees). In this case, we optimize only for the width of n_i and distribute the change in this width evenly to its descendants in a recursive manner. Further reduction of the variables is possible considering the split direction of the nodes. If a node n_i is subdivided into a set of leaf nodes by a y split, all of its children have a width equal to the width of n_i. In this case, we collapse the subtree rooted at n_i and express any constraint involving its children in terms of the width of n_i (see Figure 3.8, green subtrees). We perform such node collapses iteratively until no further collapse is possible reducing the number of variables significantly.

3.7 Evaluation

In this section we discuss the main features of our framework for several example editing scenarios. For a larger collection of editing results, please refer to the video and the

supplementary material of the corresponding publication [15]. Note that manually specifying a general grid used in these examples takes around 20 seconds (see the accompanying video). In the following results we show the final state of the active grids selected at each editing step to highlight the changes to a facade (Figure 3.9, 3.11-3.13).

Data sets. The input to our framework is a hierarchically decomposed ortho-rectified facade image. Such a decomposition can be obtained manually or in a semi-automatic manner. We evaluate our method on facade images decomposed by the semi-automatic approach of Musialski et al. [59]. We also create a shape grammar for rectified facade images taken from online repositories using Esri CityEngine and convert this grammar into a decomposition tree. We next describe the capabilities of our framework on these input facades of varying structural complexity.

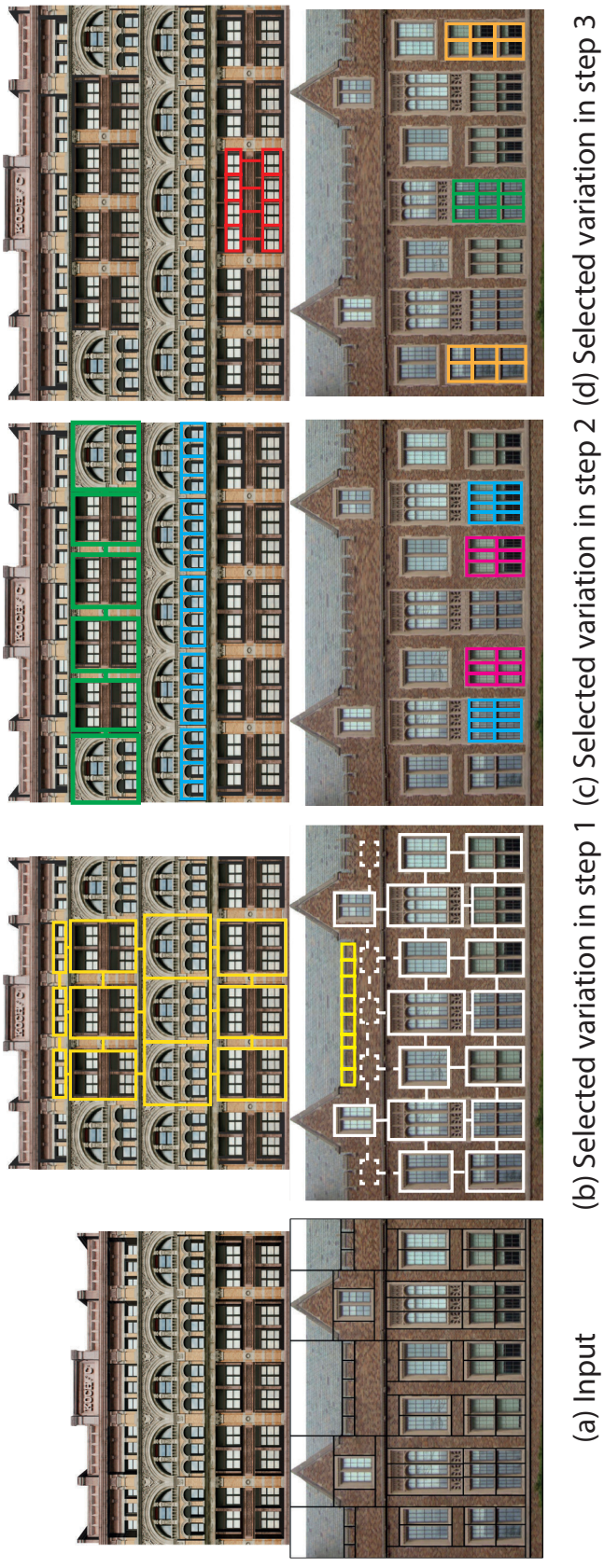


Figure 3.9 – By activating different grids (colored) at each editing step, interesting facade variations can be generated.

(a) Input (b) Editing output

Figure 3.10 – The door of the facade, that spans the bottom two rows, is assigned to the lower row, a phantom element (cyan) is added to the upper row.

Results. The input to our system is a hierarchical decomposition of a facade. However, the editing operations are not restricted by the splitting levels of this decomposition. Specifically, we allow facade elements that do not belong to the same parent element to be grouped into a grid (see Figure 3.9, row 2). This enables to jointly edit elements across different levels of the hierarchy and eliminates the dependency on the input decomposition.

Our editing framework is incremental where at each editing step, the elements of active grids are removed or duplicated. The elements of the remaining grids, on the other hand, are only scaled to respect the new size of the facade. By activating different grids at each step of this process, interesting variations of a facade can be generated, which is difficult otherwise (see Figure 3.9).

In some facades, certain elements such as a door can span multiple rows (or columns) as shown in Figure 3.10. When the elements in the rows spanned by the door are grouped into a grid, we assign the door to one of these rows resulting in fewer grid elements in the remaining row(s). In order to obtain a complete grid in such a case, we add a *phantom element* to the rows with fewer elements. The phantom element acts as a placeholder in the grid and can be duplicated or removed based on the editing operation. On the other hand, if an element is required to be inserted between two grid elements of which at least one is phantom, we simply discard this insertion (see Figure 3.9, last row).

An important feature of our framework is the support for hierarchical grids. The facade shown in Figure 3.11 consists of a grid of windows each of which is composed of a sub-grid.

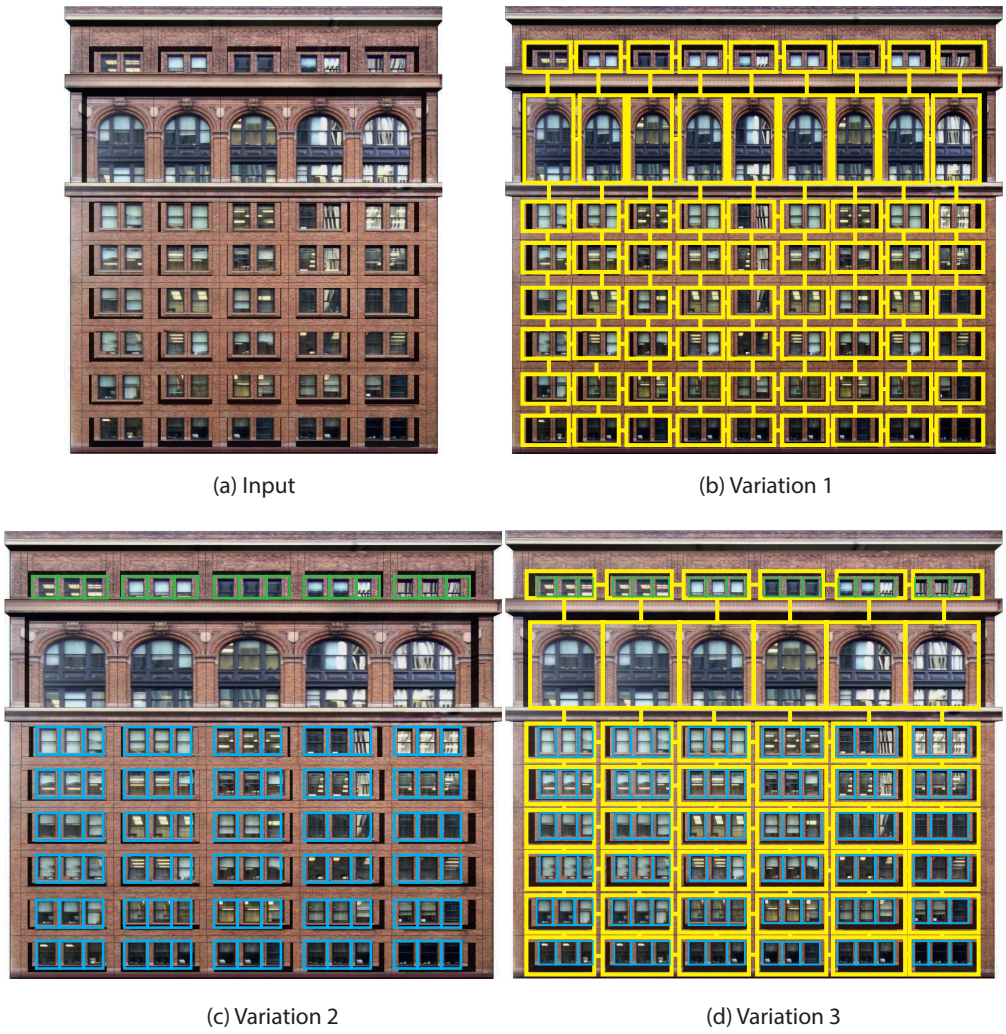


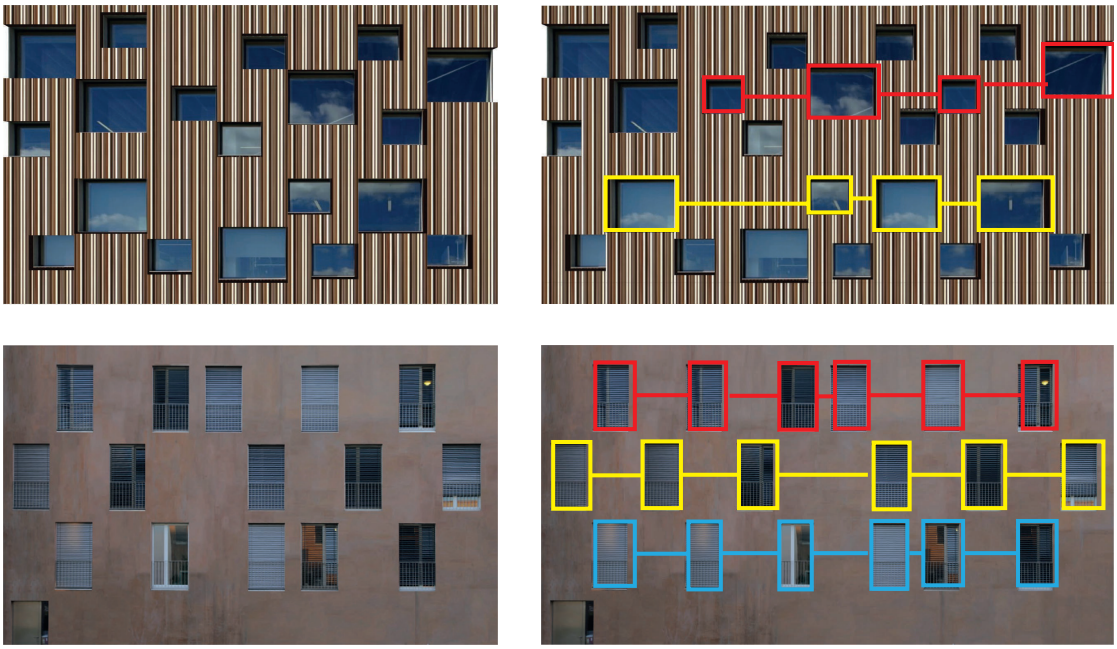
Figure 3.11 – Given the hierarchical grid structure, one can modify the main grid (yellow), the sub-grids (green and cyan), or both.

When this facade is resized horizontally, one can change the inner sub-grids of the window frames, add a new column of windows or perform both actions simultaneously. The user can explore these options by activating or deactivating the sub-grids at each editing step.

A unique feature of our approach is the notion of general grids that enable the grouping of different element types with varying spacings. For example, in Figure 3.12 one large grid consisting of different window types has been built. When the facade is resized in the vertical direction to trigger the addition of a row, any row of the grid can be duplicated. Moreover, when a new column is inserted, the corresponding window types are duplicated in each row. Even though the concept of general grids is capable of handling irregularity, some facades as shown in Figure 3.13 exhibit no dominant grid

(a) Input (b) Editing output

Figure 3.12 – Different element types are grouped in a common active grid (in green) and edited together.



(a) Input (b) Editing output

Figure 3.13 – Multiple small grids (colored) are activated to edit the facades that lack a dominant grid structure.

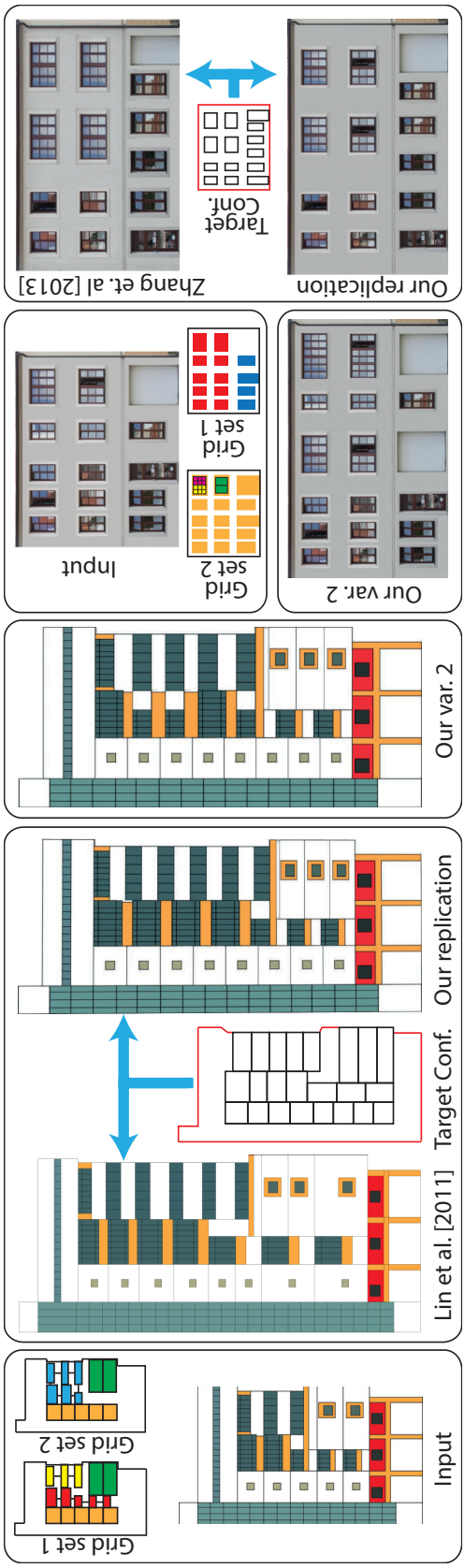


Figure 3.14 – Editing results in comparison to [47] (left) and [97] (right). Our replications of the target configurations shown in box abstraction, are generated by activating *grid set 1*. Additional outputs are produced by activating *grid set 2*.

structure. Grouping all the elements in these facades results in a grid with many phantom elements making it difficult to edit. However, editing can be performed by activating multiple small grids and preserving these local structures simultaneously.

Comparison. In this section, we provide comparisons with the methods of Lin et al. [47] and Zhang et al. [97]. We evaluate our method on two examples taken from these works and generate a replica of the same target facade configuration. We also generate additional variations to emphasize the differences between the methods (see Figure 3.14). In the retargeting framework of Lin et al. [47], reshuffling of elements in an extracted sequence is not supported. In contrast, we allow rows and columns to be inserted at each possible position in an active grid (see how small and large balconies in the red sequence are arranged in *variation 2* in Figure 3.14). A major advantage of the layered decomposition proposed by Zhang et al. [97] is the ability to move a layered element to an arbitrary new position. We achieve similar editing results by duplicating such an element at the target position and deleting the original. However, directly moving around of elements might be more intuitive for the users. Interleaving grids of different element types can be edited with the method of Zhang et al. [97] only if they have been decomposed as different layers. In that case, the user also needs to specify additional constraints to position such grids during editing. In our method, however, such elements can be grouped into a general grid and edited easily. We also support editing of hierarchical sub-structures (see how small and big window types are grouped together in *variation 2* in Figure 3.14 and edited together with their substructures).

Limitations. Even though our system is able to generate many variations of an input facade, there are several limitations we would like to address in future work. The input to our system is a decomposition of a facade with vertical and horizontal splitting lines. Such a decomposition fails to provide a tight partitioning for other polygonal and arched shapes. We represent these shapes with their axis-aligned bounding boxes. Integrating non-axis aligned splits and curved shapes will provide more plausible results for certain architecture styles with dominant curved structures.

In this work, we do not focus on smart texture synthesis, the texture of an edited facade is synthesized by duplicating, cropping, and scaling the original texture. In presence of occlusions and strong lighting variations, this results in visual artifacts. In the future, we would like to incorporate more advanced texture synthesis methods to reduce such

artifacts.

3.8 Concluding Remarks

We have presented a novel structure-preserving facade editing framework. Our approach gives direct access to shape ambiguities during editing, while avoiding the combinatorial explosion of potential editing results through an incremental process. The concept of generalized grids allows capturing both regular and irregular structures, including hierarchical configurations, which greatly expands the set of possible shape manipulations. We believe that this approach provides a novel perspective on structure-aware editing that has potential for many other geometric design tasks.

There are several interesting avenues for future work. The input to our system is a hierarchical decomposition of a facade. Conceptually it is possible to extend our method to handle layered decompositions as proposed by Zhang et al. [97] by enabling the grouping of elements at different layers into a common grid.

Although we only focused on 2D facade editing, our system can be extended to 3D by recursively decomposing 3D buildings using axis-aligned cutting planes similar to [47]. Our current framework explores structural ambiguity by supporting incremental edits and providing alternatives at each step in a sequential manner. However, defining a *structural similarity* measure between variations of an input facade and recursively clustering these variations based on this measure is an interesting research direction. Such a clustering approach, especially if built upon some properties of human perception system such as Gestalt rules [62], will enable to explore the design space more effectively. Finally, we believe extending the principles of combining continuous and discrete changes as well as neighborhood synthesis to 3D models with dominant axis-aligned structures, such as furniture, is possible and interesting.

4 Interactive Design of Probability Density Functions for Shape Grammars

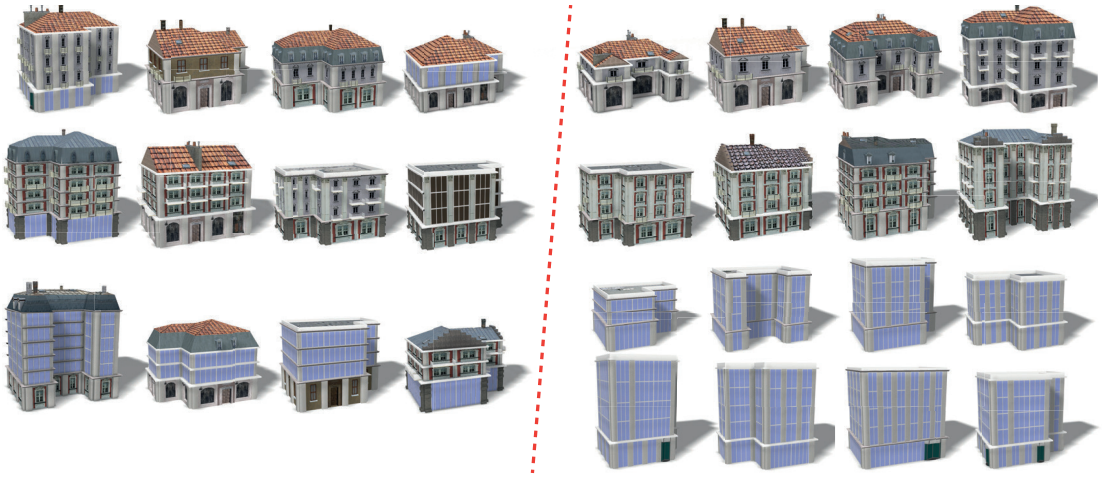


Figure 4.1 – (Left) Random models generated from a probabilistic building grammar. Although these models are visually plausible, they do not comply with a design scenario which also requires architectural plausibility, i.e. matching styles of ground floors, upper floors, and roofs (B1, see Table 4.2). (Right) Our framework takes user specified preference scores as input and learns a new model probability density function (pdf) which samples models (with consistent style) proportionally to their predicted preference scores. In this design scenario, office buildings received a higher preference score.

In the previous chapter, we discuss about the structure-aware editing approach to generate structured variations. We will now investigate another modeling technique, the procedural modeling paradigm, which is capable of automatically generating a huge collection of shape variations conforming to a global structure e.g. buildings, trees, furniture, airplanes, bikes, etc. Despite its powerful modeling capability of procedural modeling techniques, it is difficult to control the generation output, in particular the space of generated models and the generation likelihood of each model. This remains a core challenge in this modeling paradigm.

We present a framework that enables a user to interactively design a probability density function (pdf) over such a shape space and to sample models according to the designed pdf. First, we propose a user interface that enables a user to quickly provide preference scores for selected shapes and suggest sampling strategies to decide which models to present to the user to evaluate. Second, we propose a novel kernel function to encode the similarity between two procedural models. Third, we propose a framework to interpolate user preference scores by combining multiple techniques: function factorization, Gaussian process regression, auto-relevance detection, and l_1 regularization. Fourth, we modify the original grammars to generate models with a pdf proportional to the user preference scores. Finally, we provide evaluations of our user interface and framework parameters and a comparison to other exploratory modeling techniques using modeling tasks in five example shape spaces: furniture, low-rise buildings, skyscrapers, airplanes, and vegetation.

4.1 Foreword

Procedural modeling using grammars is a very effective tool to generate a large variety of similar models. Grammars have been successfully used for modeling vegetation, buildings, building interiors, streets, sea shells, furniture, feathers, etc. Even though grammars are a great tool for modeling, generating good grammars is difficult and requires some programming skills. After talking to many users of the procedural modeling software Esri CityEngine , we can identify two groups of users of grammar-based procedural modeling. Non-expert users such as architects, urban planners, and regular artists without training in computer science, mainly use existing grammars, possibly with minor customizations. Expert users, typically technical artists or computer scientists with some art background, can generate new grammars from scratch. A common problem that both groups of users face is the difficulty to control the distribution of models generated by a grammar. In

our paper we use the term *shape space* to describe the set of all models that can be generated by a grammar. Further, the *probability density function* (pdf) of a grammar determines how likely a model is to be generated. We propose new interactive interfaces and techniques to design the pdf of a given grammar, without editing the grammar rules directly. To motivate our research we discuss three example workflows of non-expert and expert users.

The goal of the first workflow is to generate a single interesting model. A user, e.g. an architect, might like to navigate the shape space of a grammar to get a design idea for a new building. Currently, the main solution is to randomly regenerate a model many times. Following pioneering work in exploratory modeling, e.g. [51, 85], we would like to give the user the ability to navigate the shape space by designing a suitable pdf with high density in the neighborhood of the previously liked models.

The goal of the second workflow is to customize the pdf of an existing grammar (for non-expert users) or to fine tune a grammar written by an expert user for large-scale modeling, e.g. a complete city. For example, a user might want to control the distribution of building heights, building styles, building functions (e.g. residential vs. commercial), and the assets (meshes and textures) used for windows, doors, and ornaments. In simple cases these distributions can be controlled by a single parameter in the grammar, but in many cases the user will be interested in influencing aspects of the distribution that involve more than one parameter. Due to the context-free nature of most shape grammars, this is difficult to encode and would need major changes to the grammar rules. For example, it requires some work to encode that tall buildings should be predominantly gray and small buildings should be mainly brown.

The goal of the third workflow is to write a grammar that can generate a large variety of models. An expert user might start with a deterministic grammar that is able to generate a single high quality model. In a subsequent step, the user can introduce initial randomness to generate some minor variations, most of them having high quality. As the user adds more and more rules and randomness to the grammar, the probability of generating low quality models increases due to the trade-off between model variety and model quality. The reason for this degradation is again a limitation of context-free rules. The rules make design choices without global context, and it is difficult to coordinate design choices made by different rules. In Figure 4.1 we illustrate two example problems: matching the style of ground floor and upper floors, and matching the roof style of a building with its facade style. The price for having a large shape space to choose from

will typically be a larger percentage of undesirable models. Our work enables an expert user to focus on writing simpler rules encoding the structure of models and then using our proposed framework to model a pdf that eliminates most undesirable models.

4.2 Contributions

In this chapter, we makes the following contributions to the state of the art:

- On the application side, we are the first to extend the concept of exploratory modeling from selecting a single model to interactively designing a pdf for a procedural shape space.
- On the user interface side, we propose strategies to display, sort, and sample models to enable the user to quickly provide preference scores.
- On the technical side, we integrate concepts from machine learning with context-free shape grammars. First, we propose a novel kernel function to encode the similarity between two procedural models. Second, we propose a framework to interpolate preference scores given by a user combining multiple techniques: function factorization, Gaussian process regression, autorelevance detection, and l_1 regularization. We show that our framework leads to better results than the kernel density estimation framework proposed by Talton et al. [85]. Third, we propose the first algorithm to automatically generate a new grammar that approximates the target pdf well.

4.3 Overview

4.3.1 Framework Overview

The input to our framework is a shape grammar. The default values of the rule probabilities and rule parameters defined by the grammar impose a default density function over the procedural space S from which output shapes can be sampled. Our goal is to model a new density function that reflects the user preferences as closely as possible and then to generate a new grammar that samples shapes according to this new density function.

Our framework has the following major components: 1) A user interface that enables a user to provide preference scores for selected models in the shape space of a given input

grammar (see Section 4.4). 2) A regression algorithm that interpolates the user defined preference scores (see Section 4.5). 3) An algorithm to generate new models according to the derived pdf (see Section 4.6).

4.3.2 Definitions

We use attributed, stochastic, context-free shape grammars that generate models by shape replacement. Each shape has a list of attributes and a label from the set of non-terminal symbols NT or from the set of terminal symbols T. While the number of attributes can vary depending on the grammar, we require at least the following: an oriented bounding box, called scope, a polyhedral mesh that is transformed to fit inside the scope, and a material identifier. We use an example grammar, called *toy grammar*, shown at the end of this subsection to illustrate various concepts in this paper. Random samples of this grammar are shown in Figure 4.2. A shape grammar is defined as a tuple:

$$G = \langle NT, T, \omega, P, \Theta \rangle, \tag{4.1}$$

where $\omega \in NT$ is the starting symbol. The set of all symbols is denoted by $SYM = NT \cup T$. P is a set of context-free rules of the form

$$id_i \; prob_i : shape_i \rightarrow ShapeOp_i, \tag{4.2}$$

where $shape_i \in NT$ and $ShapeOp_i$ is a sequence of shape operations that generates replacement shapes $(succ_{i1}, \ldots, succ_{ik})$ with $succ_{ij} \in SYM$. The number j is called the *child index* and it can be used to identify a particular shape among the k replacement

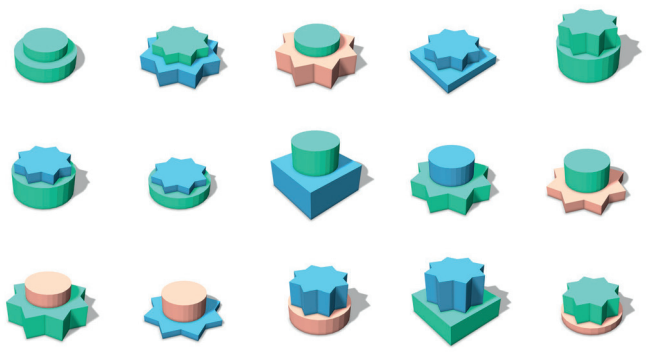


Figure 4.2 – Random samples of the toy grammar in Section 4.3.2.

shapes. The probability of the rule being selected is $prob_i$ so that the probabilities for all rules that have $shape_i$ on the left hand side need to sum up to one. The rule id id_i is generated automatically and mainly used so we can refer to rules in the paper. The parameter vector Θ includes all information about probabilistic choices available in the grammar. These are a) the rule probabilities and b) all parameters of the distributions used to sample random values in shape operations, e.g. the minimum and maximum values of the uniform distributions to sample the heights h_1 and h_2 in the toy grammar.

There are quite a few details about how shape operations could be specified, but these details are not important for the core contribution of the paper. In our examples we use grammars that are compatible with CityEngine and we use shape operations that modify the scope of a shape, e.g. via translation, rotation, and scaling, set the mesh of a shape (see the **i**() operation in rules id_2 to id_4), split shapes into multiple smaller shapes, or combine multiple shapes.

In the toy grammar the rule id_1 stacks two shapes with height h_1 and height h_2 on top of each other to yield a top shape and a bottom shape. The rules id_2, id_3, and id_4 generate either a box, cylinder, or star geometry with equal probability 1/3. id_5 is an example for a redundant rule. Finally id_6 to id_8 select a color for the generated geometry.

Toy Grammar

```
1  NT = {MASS, MESH, MESH2}
2  T = {TerminalMesh}
3
4  attr h1 = rand(1, 5)
5  attr h2 = rand(1, 5)
6
7  id1 1   : ω       → split(y) { h1: MASS | h2: MASS }
8  id2 1/3 : MASS    → i("box") MESH
9  id3 1/3 : MASS    → i("cylinder") MESH
10 id4 1/3 : MASS    → i("star") MESH
11 id5 1   : MESH    → MESH2
12 id6 1/3 : MESH2 → setMaterial("orange") TerminalMesh
13 id7 1/3 : MESH2 → setMaterial("blue") TerminalMesh
14 id8 1/3 : MESH2 → setMaterial("green") TerminalMesh
```

This grammar already illustrates a limitation of context-free grammars. For example, it is not possible to modify the parameter vector Θ to ensure that the bottom shape has the color orange and the top shape has a random color. That requires changing the structure of the grammar (the rules in P) by duplicating and changing the rules id_2 to id_8. If there were 5 shapes stacked on top of each other it would already become unmanageable to encode the probabilities for various geometry and color combinations. Unfortunately, any context-free grammar requires an exponential number of rules to encode general

Figure 4.3 – Main components of the user interface. (a) Display: Models (sampled from the current pdf) are shown with their predicted preference scores (orange) and prediction uncertainties (blue, see Section 4.5.3). (b) Sorting: Models are sorted by the similarity to the selected model (green). (c) Selection and assignment: Multiple models can be selected and assigned the same preference score at once.

joint probability distributions of multiple variables. Nevertheless, we can automatically generate context-free grammars approximating a given pdf. The grammar might be too complex to be human readable, but it is compatible with existing derivation engines and can be used for fast model generation.

4.4 User Interface

In the following we will describe the user interface and the interaction possibilities and then give the details for the technical realization in Section 4.5. The main idea of the user interface is to enable the users to provide simple feedback about their preferences on a set of shapes and then use this feedback to compute a preference function that can assign a preference score to each model in the procedural shape space. The density function is the normalized preference function. Two important characteristics of the user interface are how quickly a user can provide a preference score and how much knowledge is gained by scoring the shown models.

The user interface is shown in Figure 4.3. Models sampled from the procedural space are scaled and organized in one or two regular grids. Our framework not only predicts the preference score of each model but also can estimate the prediction uncertainty (see Section 4.5.3). Thus, for each model we optionally show its predicted preference score by an orange bar (in the range 0-100) and the uncertainty of the prediction in blue. Note that the range of scores will be normalized later so that only the ratio between scores is

important and not their absolute values.

The user can choose how many models are shown in the grid(s). We typically use smaller grids, e.g. 4×4, to rate models and larger grids, e.g. 10×10, to check the results. The user can also zoom and pan to see individual models in more detail.

Further, the user can decide how the models are generated to populate the grid. There are five choices: a) uniform, b) grammar, c) density function, d) uncertainty, and e) already ranked. Uniform sampling selects candidates that are as dissimilar as possible using furthest point sampling among a set of candidates generated by the grammar. The grammar option simply samples models using the pdf of the original grammar. When the user selects the density function option we sample one grid from a pdf that is proportional to the preference function and a second grid to sample from the complementary pdf (100 minus the preference score and then normalized). Showing these two grids helps us to identify models that are undesirable, but have a predicted high preference value and models that are desirable, but have a predicted low preference value. The option d) allows sampling models according to their prediction uncertainty. The last option e) is used to review already ranked models to check for mistakes.

The user can also choose different sorting strategies. The models in the grid can be sorted according to their preference score, randomly, or according to an individual feature (see Section 4.5.1 for a description of features). The user can also select a set of models and sort the remaining models according to the highest similarity score when comparing to the selected models. The user is able to use standard selection techniques to provide preference scores for selected models. After changing scores for one or more models the user can trigger an update to re-estimate the density function and resample models shown in the visualization. This process is repeated until the user is satisfied with the modeled density function.

More complex density functions can be modeled as the normalized product of individual preference functions, also called *factors*. Typically, these factors describe the users preference about particular aspects of the shapes in the shape space. For example, for the simple grammar in Figure 4.2 the user might want to specify her preference for the color and the geometry of the models separately. To enable the user to work with factors, we provide options to create, delete, and name preference functions (factors). Further, the user can set an active preference function to indicate which preference function she wants to work with.

4.5 Learning the Probability Density Function

The user interface described in the previous section enables the user to assign preference scores to selected models of the shape space. We now describe our technical solution to obtain a probability density function for the complete shape space using the following steps: 1) We map procedural models into a feature space (see Section 4.5.1) in which we assume preference scores are smoothly varying. 2) We define an overall preference function (see Section 4.5.2) by combining individual preference functions (factors). 3) We use a non-linear regression technique to interpolate the user preference scores specified for a set of models to the rest of the shape space. In this part we explain the regression model, the regularization, and the estimation of hyperparameters of the kernel function using automatic relevance detection (see Section 4.5.3).

4.5.1 Features

Finding a good function that maps a procedural model m to a feature space $\mathcal{X}$ is a challenging problem. We initially experimented with geometric features, but realized that such features need to be specifically designed for each grammar separately. For parametric models, e.g. Talton et al. [85], one can simply use the model parameters as features directly and obtain good results. While grammar parameters have been used as features previously [79], this approach only works for simple grammars where all models have the same structure. For more interesting grammars, each generated model can be encoded by its derivation tree so that the fundamental problem of feature design is to encode the discriminative properties of the derivation tree (see Figure 4.4 for an example).

The feature mapping scheme in our framework is inspired from the tree kernel in natural language processing [14]. In specific, we count the number of occurrences of each potential tree path in the derivation tree and use these numbers to map the procedural model to the feature space. A path in the derivation tree is identified by the sequence of edges. We label each tree edge by a tuple consisting of the associated rule and the *child index* (see Section 4.3.2). For example, for the derivation tree in Fig. 4.4 we can observe a path $(\langle id_1, 2\rangle, \langle id_4, 1\rangle)$. For a detailed discussion of this mapping scheme, we refer the audience to the corresponding publication [16]. In Figure 4.4 we show the extraction of selected features from a derivation tree.

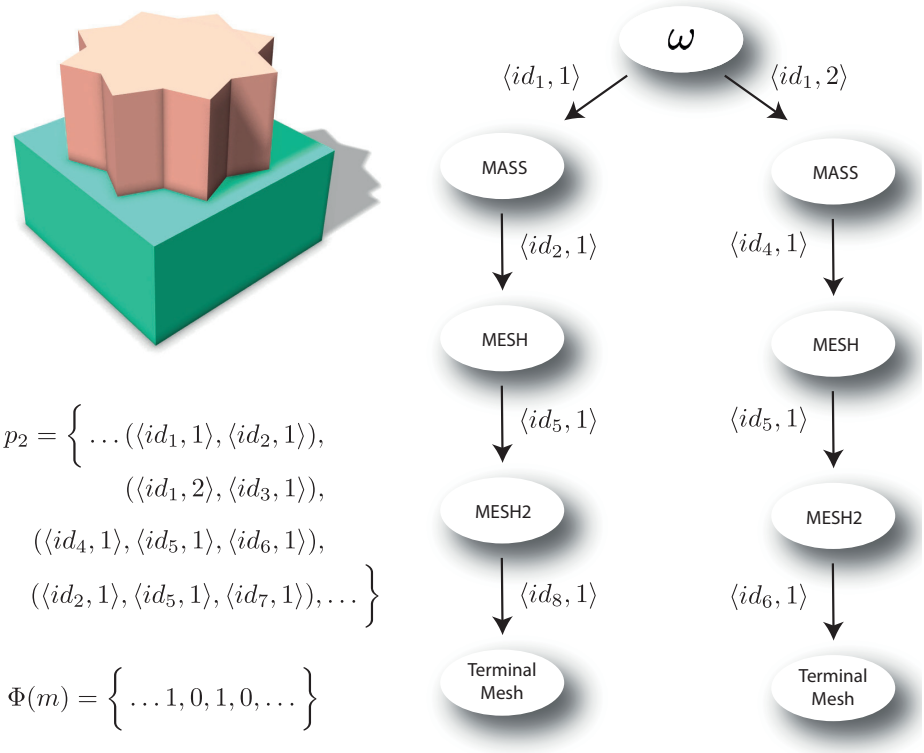


Figure 4.4 – Feature mapping. A model m generated from the toy grammar is shown together with its derivation tree. We show selected features as possible tree paths with effective length 2 (p_2) and the mapping of the shown model into feature space $\Phi(m)$. The mapping assigns a value v if the corresponding path exists v times in the derivation tree.

Since there can be a large number of uniquely identifiable paths, we need to select a useful subset of such paths. We make the important observation that rule sequences of different lengths have varying discriminating power. While shorter sequences are capable of distinguishing models via the presence of more general properties (e.g. an orange box versus an orange cylinder), longer sequences separate more specific models (e.g. an orange box on the bottom versus an orange box on top). After analyzing many useful preference functions, we concluded that most basic concepts can be explained by short paths. Therefore, we start out using only paths of up to length k. We then gradually add longer paths as features if the current feature set becomes incapable of discriminating shapes that are assigned different preference scores by the user or the user requests more complex features. In Figure 4.8 we evaluate the effect of this maximum path length k.

4.5.2 Preference Function Factorization

In our system we allow the user to either model a single global preference function or to model the preference function as a product of individual preference functions, called *factors*. The idea of using multiple preference functions is that the user might want to specify preferences about particular aspects of the models instead of the models as a whole. Examples for factors would be the preference for certain materials, roof shapes, mass models, or window styles. A similar concept, visual attributes, has been employed in computer vision for various tasks such as object recognition [25].

As an example, consider the following two factors for the toy grammar from Section 4.3.2. 1) *Color*: The user prefers models that have the same color for the bottom and the top mesh with preference 100 for orange and 50 for green and blue. 2) *Geometry*: The user is interested in mass models where the bottom mesh is a box with preference 100, a cylinder with preference 50, and a star with preference 0. In Figure 4.5 we show preference scores for the individual preference functions (factors) as well as their combination. A factor can involve one or multiple features. Most importantly, in most cases the sets of features necessary to evaluate different factors are not identical. This is what makes learning with individual factors more efficient compared to specifying a single preference function.

Specifically, let m be a procedural shape which can be mapped into a D-dimensional feature space $\mathcal{X}$ by $\Phi(m) = \mathbf{x} = [x_1, x_2, \dots x_D]^T$ (in the rest of the paper we will use m and $\mathbf{x}$ interchangeably). We model the user preference function, $u(m) = u(\mathbf{x})$, in this space as the product of K factors:

$$u(\mathbf{x}) = \prod_{i=1}^{K} u^i(\mathbf{x}). \tag{4.3}$$

To train a factor u^i, the user provides preference scores for a set of models, considering only the corresponding semantic aspects of that factor. We then interpolate these scores to the rest of the shape space. We accomplish this task by using a kernel-based Bayesian regression technique called Gaussian Process Regression (GPR).

4.5.3 Gaussian Process Regression

GPR assumes a Gaussian Process prior over the preference function u^i. Thus, any finite set of n observations $[u^i(m_1), \dots, u^i(m_n)]^T$ can be considered as an n-dimensional point

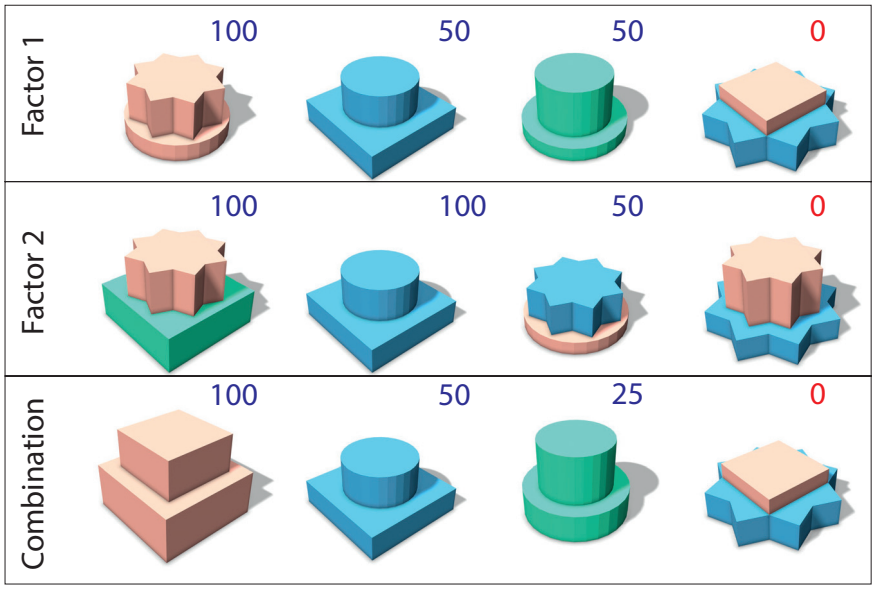


Figure 4.5 – A factorized preference function for the toy grammar. Two factors are trained separately and later combined to obtain the overall preference function. The factors are described in the text.

sampled from an n-variate Gaussian distribution. This method can work well with small training sets, which is necessary for our active learning framework where we expect reasonable results starting from the first iteration. For a more detailed discussion of GPR, we refer the reader to the work of Rasmussen and Williams [71].

Similar to parameterizing a Gaussian distribution by its mean and the covariance matrix, a Gaussian Process prior over u^i is specified by a mean function indicating the prior bias of u^i and a kernel function $k(m, m')$ which specifies how similar $u^i(m)$ and $u^i(m')$ are. In our framework, we assume a zero-mean Gaussian Process prior and learn the kernel function from user input. This initially assumes that all models are not wanted, an assumption that was successfully employed in similar contexts, e.g. music ranking [67].

Prediction. Given a set of n models $\mathbf{m} = [m_1, m_2, \ldots, m_n]^T$ and the corresponding user preference scores, $\mathbf{u}^i = [u^i(m_1), u^i(m_2), \ldots, u^i(m_n)]^T$, our goal is to predict the preference $u^i(m^*)$ of any other model m^*.

Assuming $u^i(m)$ has a zero-mean Gaussian Process prior with the kernel function $k(m, m')$, $[\mathbf{u}^i, u^i(m^*)]^T$ has a joint $(n+1)$-variate zero mean Gaussian distribution with the following

covariance matrix:

$$\mathbf{C}_{n+1} = \begin{bmatrix} k(m_1, m_1) & \dots & k(m_1, m_n) & k(m_1, m^*) \\ & & \vdots & \\ k(m_n, m_1) & \dots & k(m_n, m_n) & k(m_n, m^*) \\ k(m^*, m_1) & \dots & k(m^*, m_n) & k(m^*, m^*) \end{bmatrix}.$$

The conditional distribution $p(u^i(m^*)|\mathbf{u}^i)$ is also a Gaussian distribution, which leads to a closed form solution for $u^i(m^*)$ defined at the mean of this distribution as:

$$u^i(m^*) = \mathbf{k}^T \mathbf{C}_n^{-1} \mathbf{u}^i, \tag{4.4}$$

where $\mathbf{C}_n$ is the top-left $n \times n$ block of $\mathbf{C}_{n+1}$ and

$$\mathbf{k} = \begin{bmatrix} k(m_1, m^*) \\ \vdots \\ k(m_N, m^*) \end{bmatrix}.$$

In addition to predicting $u^i(m^*)$, GPR also provides the *prediction uncertainty* which measures the confidence of the prediction. This uncertainty is defined as the variance of $p(u^i(m^*)|\mathbf{u}^i)$ and calculated as follows:

$$\sigma^2(m^*) = k(m^*, m^*) - \mathbf{k}^T \mathbf{C}_n^{-1} \mathbf{k}. \tag{4.5}$$

We utilize the variance values in our sampling strategy based on prediction uncertainty (see Section 4.4) to determine the models presented to the user.

Kernel function. Given a kernel function, we interpolate the user preference scores from a set of shapes to the procedural space using GPR. The choice of the kernel function heavily influences the results of this regression process. Instead of using a fixed kernel function, we propose to use a family of parametrized kernel functions and learn the kernel parameters from user input.

To encode the similarity between procedural models m and m', we use an anisotropic

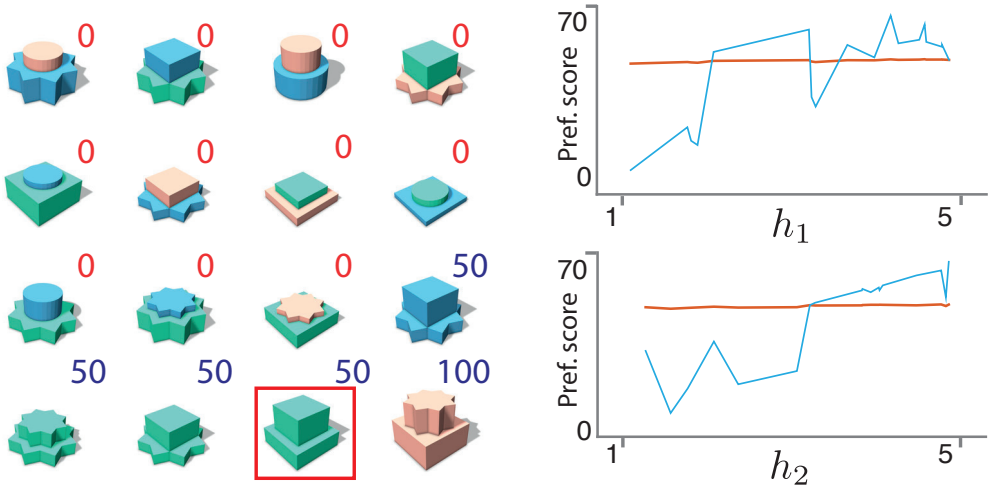


Figure 4.6 – Auto-relevance detection (ARD). We assign preference scores for the 16 models on the left to train the "Color" factor of the toy grammar (Section 4.5.2). Then we take the green box model highlighted by a red rectangle and create a lot of variations by changing the heights h_1 and h_2. The plots on the right show the predicted preference scores for h_1 and h_2 varying from 1 to 5. The preference scores learned with ARD are shown in orange and without ARD in cyan. As the orange lines remain almost constant when h_1 and h_2 change, this suggests that those two parameters are irrelevant to the preference scores.

kernel function k defined over the D-dimensional feature space $\mathcal{X}$ as:

$$k(m, m') = \theta_0 \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \theta_d (x_d - x'_d)^2 \right) + \beta^{-1} \delta_{mm'}, \tag{4.6}$$

where x_d and x'_d are the features of m and m' respectively. $\beta^{-1}\delta_{mm'}$ is a small added noise to ensure positive-definite covariance matrix where $\delta_{mm'}$ is the Kronecker delta. The kernel hyperparameters $\theta_d, d \in [1, D]$ are adapted every time a new user preference score is specified. They are feature weights indicating the relevance of the corresponding features. Intuitively, features with small weights do not have a high influence on the preference scores. We learn these parameters by automatic relevance detection via maximizing the likelihood of the training data as explained next.

Automatic relevance detection. Typically, the influence of a feature depends on the preference function a user wants to model. Thus, instead of assigning a fixed θ_d to each feature, we adapt the hyperparameters of the kernel function based on user input. Additionally, our goal is to discard the irrelevant features by assigning them a

0-weight. We achieve this goal by optimizing for the hyperparameters $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots \theta_D)$ that maximize the log likelihood of the training samples:

$$\log p(\mathbf{u}^i|\boldsymbol{\theta}, \mathbf{m}) = -\frac{1}{2}\log|\mathbf{C}_n| - \frac{1}{2}\mathbf{u}^{iT}\mathbf{C}_n^{-1}\mathbf{u}^i - \frac{n}{2}\log(2\pi). \tag{4.7}$$

In order to avoid overfitting, we add a Lasso regularization term and minimize the following energy function:

$$E(\boldsymbol{\theta}) = -\log p(\mathbf{u}^i|\boldsymbol{\theta}, \mathbf{m}) + \lambda \sum_{d=1}^{D} |\theta_d|. \tag{4.8}$$

We define the hyperparameters to be non-negative, leading to $|\theta_d| = \theta_d$. Instead of defining constraints to avoid negative θ_d, we optimize for $\log(\theta_d)$ using the Conjugate Gradient method. The additional Lasso term also favors as few non-zero hyperparameters as possible resulting in the assignment of 0-weights to irrelevant features.

When training two attributes of the toy grammar (see Section 4.5.2), the block heights (h_1 and h_2) are irrelevant to the preference scores. This can be detected by our framework, as illustrated in Figure 4.6.

4.6 Generating Models According to a Probability Density Function

After learning the preference function $u(m)$, the next step is to generate procedural models with a pdf proportional to $u(m)$. We provide three options for this task including 1) Parameter learning: modifying the grammar parameters without changing the rule structure. 2) Structure learning: generating a new context-free grammar by changing both rule structure and rule parameters. 3) Rejection sampling: standard rejection sampling [5] as a post-process on generated models.

Parameter learning is very simple, but not very flexible. Thus the learned pdf does not match the target pdf well in most cases. Rejection sampling produces results exactly according to the target pdf, but it is slow and not compatible with existing grammar derivation engines. Structure learning is the default solution of our framework. It approximates $u(m)$ well and sampling can be done by the computed context-free grammar directly.

Parameter learning. This approach keeps the rule structure of the input grammar and only modifies rule parameters, specifically rule probabilities. We use the standard histogram-based estimation of rule probabilities of probabilistic context-free grammars (c.f. [30]). This naive solution does not perform well as shown in Table 4.3. The main problem is that it ignores the dependency between rule choices. For example, the choice of table bases may depend on how many legs this table has.

Structure learning. We propose this second option to deal with the aforementioned drawback. We use a *splitting operation* to split the original grammar and then train a mixture of grammars. While this generates a context-free grammar with more rules, it gives us the flexibility to encode conditional dependencies between rule choices.

We propose to use a splitting operation that is inspired by the concept of parent annotation in natural language processing [30]. The splitting operation splits a symbol in the original grammar into multiple different symbols by indexing it with the parent *edge labels* (see Section 4.5.1). The splitting operation therefore changes rules where the symbol appears on the right hand side and it duplicates all rules having the original symbol as left hand side. For example, in Figure 4.13 a V-shape block at the ground level of the Skyscraper grammar can obtain a different symbol from a V-shape block in the upper level. The splitting operation does not change the shape space, but it increases the degrees of freedom to manipulate its pdf. Recursively splitting every symbol in the original grammar leads to an exponential growth in the number of rules. Thus, we only split the symbols involved in the relevant features detected by the learning process in Section 4.5.3.

After performing the described splitting operations we train a new grammar G' as the combination of K component grammars G'_i. (K is determined by the algorithm described later.) In this mixture, a component grammar G'_i is selected with a probability α_i:

$$\alpha_i : G' \rightarrow G'_i.$$

We start with a training set $T = (m_1, m_2, \ldots m_N)$ sampled from the target pdf $u(m)$ (using rejection sampling). We then find a set of hyperparameters $\boldsymbol{\eta}$ consisting of α_i and the grammar parameters of G'_i to maximize the log likelihood of the training set $\mathcal{L}(T : G') = \sum_{i=1}^{N} \log p(m_i|G')$. The value of $p(m|G')$ for generating a model m using G'

is a weighted combination of the pdfs from all component grammars G'_i:

$$p(m|G') = \sum_{i=1}^{K} \alpha_i p(m|G'_i). \tag{4.9}$$

Similar to learning Gaussian Mixture Models (GMMs), we optimize for $\boldsymbol{\eta}$ using an Expectation-Maximization approach. Both E-step and M-step are identical to those in the GMM case, except that in the M-step, we train grammar parameters instead of Gaussian parameters. In particular, in the E-step, we find the affinity of G'_k to each model m_n. This affinity measures how likely m_n has been generated from G'_k

$$w_{kn} = p(G'_k|m_n, \boldsymbol{\eta}) = \frac{\alpha_k p(m_n|G'_k)}{\sum_{i=1}^{K} \alpha_i p(m_n|G'_i)}. \tag{4.10}$$

In the M-step, we first calculate the sum of affinity of each component grammar $N_k = \sum_{n=1}^{N} w_{kn}$. This affinity sum can be considered as the effective number of models that have been generated from G'_k. We can then update the mixture weights as follow:

$$\alpha_k^{new} = \frac{N_k}{N}. \tag{4.11}$$

To initialize the grammar mixture, we cluster the training set T using the kernel function learned previously, and train one grammar for each cluster using the aforementioned histogram-based method. To compute the clustering, we build a neighborhood graph, where two models are connected if their pairwise difference is smaller than a threshold. The pairwise difference between models is derived from the learned kernel. The threshold is set to 0.001 of the maximum difference. We then iteratively look for the model with the most neighbors, put it together will all its neighbors in a new cluster, and remove all elements of the new cluster from the graph.

Splitting symbols and clustering the training set significantly reduces the number of rules in our design scenarios. Nonetheless, for an artificially designed worst case preference function, an exponential growth of rules (see Section 4.3.2) cannot be avoided.

Rejection sampling. This algorithm does not compute a new grammar, but instead modifies the sampling as a post-process. For each generated model m, we calculate its generation pdf $p(m)$ by multiplying the associated rule probabilities and the pdf of parameters of shape operations (e.g. floor height). This model is accepted with the rate $\frac{u(m)}{Cp(m)}$, where the constant C is the upper bound of $\frac{u(m)}{p(m)}$ found empirically by sampling a

large amount of models in a preprocessing step. Since rejection sampling is a post-process, it can be used in conjunction with the original grammar, as well as grammars generated by parameter learning or structure learning.

4.7 Evaluation

Example grammars. We designed four grammars for our tests: Furniture, Building, Skyscraper, and Airplane. To compare to previous work we also use the Weber & Penn parametric tree model implemented in Arbaro[1]. Table 4.1 shows some statistics describing the complexity of the grammars and the parametric model.

	N_Θ	N_{SYM}	N_P
Furniture	50	38	75
Building	26	80	122
Skyscraper	39	27	61
Airplane	8	44	26
Tree	73	N/A	N/A

Table 4.1 – The table shows the number of parameters (N_Θ), the number of symbols (N_{SYM}) and the number of rules (N_P) for our input grammars and the parametric tree model.

Implementation. Our framework is implemented in `C++` and we used a MacBook Pro with Intel i7 2.6 Ghz CPU for our tests.

Design scenarios. We generate a set of design scenarios with varying complexity shown in Table 4.2. For each scenario, we define the ground truth pdf by labeling a pre-generated set of up to 5000 models. In order to validate our results, we use Jensen-Shannon divergence (also known as information radius) [50] to compare the density functions designed using our interface and the ground truth density functions. We additionally measure the correlation between these two density functions and provide the correlation scores in Appendix B. These values are computed based on the pre-sampled models.

We show the outputs of our design scenarios for the Skyscraper grammar in Figure 4.13, the Airplane grammar in Figure 4.14, and the Furniture and the Building grammar in

[1] http://arbaro.sourceforge.net

	Preference scores
F1	Valid tables with one leg (70), two legs (20) and four legs (10), non-standing tables (0)
F2	Round top & light wood tables (60), rectangular top & dark wood tables (40), others (0)
F3	Valid tables with steel materials (70) and wooden materials (30), non-standing tables (0)
F4	Valid tables with round top (70) and rectangular top (30), non-standing tables (0)
B1	Building styles: office (40), R1 (20), R2 (20), R3 (20), mixed styles (0)
B2	Big building (5-6 floors)& L-shape (50), small building (2-3 floors) & rectangular shape (50), others (0)
B3	Building shapes: L-shape (60), rectangular shape (40), others (0)
B4	Building size: big building (80), small building (20), others (0)
T1	Plausible tree (100), non-plausible tree (0)
S1	Skyscrapers with all rectangular blocks (60), V-blocks (20), cylindrical blocks (20), mixture of block types (0)
S2	Skyscrapers with rectangular base (100), cylindrical base (50), V-base (0)
A1	Old-style airplanes (Biplane or Fokker) (100), modern airplanes (commercial, transport and jet fighter) (50), airplanes with non-matching components (0)

Table 4.2 – Design scenarios for Furniture (Fi), Building (Bi), Skyscraper (Si), Airplane (A1) and Weber & Penn trees (T1). Valid tables are tables that stand by themselves. For aesthetic reasons, we also require legs and bases to match. Building styles are defined as follows. Office: glass windows, glass door and flat roof. R1: residential blocks with bright wall colors, Paris-like windows, ground floor shops. R2: residential blocks with simple windows and doors. R3: residential blocks with old-style windows and doors. Example models with their preference scores are given in Appendix A.

Figure 4.12. Each design task is represented by a 4×4 grid of models sampled from the designed density function. Bigger grids of size 10×10 can be found in Appendix B.

Evaluation of the user interface. We provide multiple tests to evaluate the performance of our framework. In the first test, we evaluate the four different sampling strategies (see Section 4.4) used to sample the models to display. The results in Figure 4.7 (a-b) empirically show that sampling from the current pdf gives good convergence rate and we choose this as the default strategy. We next evaluate the effect of the batch size, i.e. the number of models a user ranks (see Figure 4.7 (c-d)). In each step, we show two grids of size $n \times n$, one sampled from the current pdf and the other from the complementary pdf, query for the preference scores, recompute the pdf and then resample the models to show in the grids. There are two aspects we consider when choosing a batch size. First, the learning efficiency: we can see from our experiments that smaller

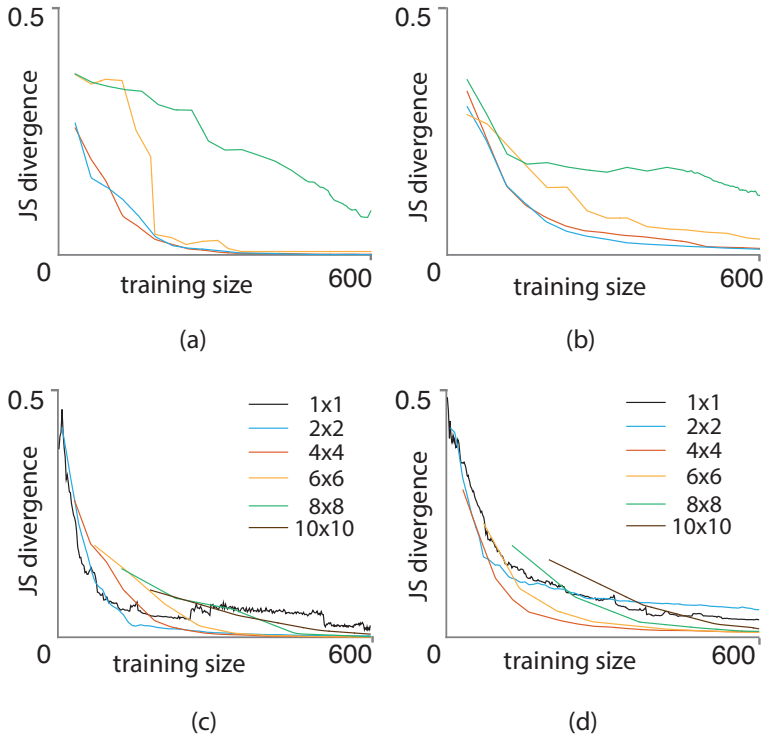


Figure 4.7 – (a) - (b) Comparison of different sampling strategies for the factor F1 (a) and B1 (b): sampling randomly (red), uniformly (green), by prediction uncertainty (orange) and based on the current pdf (cyan). (c) - (d) Comparison of sampling batch size for the factor F1 (c) and B1 (d) when using two grids sampled from the current pdf and the complementary pdf of size $n \times n$. Note that these curves start at different points due to different numbers of training data inserted in the first iteration.

batch sizes lead to more informative samples. Second, the user interface speed: in our interface, the user can quickly rank multiple models without recomputing the pdf (using a combination of sorting and selection). This favors larger batch sizes. Considering both aspects, we propose that showing two 4×4 grids to the user is a good trade-off. In the third test, we show the benefit of factorizing the preference function. We compare a factorized preference function and non-factorized preference function in Figure 4.11. Using a factorized preference score leads to much better convergence.

Evaluation of the feature design. In Figure 4.8 we evaluate the parameter k (path length) (see Section 4.5.1) on the Furniture grammar (Figure 4.8a, scenario F4) and the Skyscraper grammar (Figure 4.8b, scenario S2). For each scenario, we obtain 4 sets of features associated with paths in the production trees with maximum lengths of 1, 2, 3 and 4. While the convergence does not vary significantly in F4, the set of features with

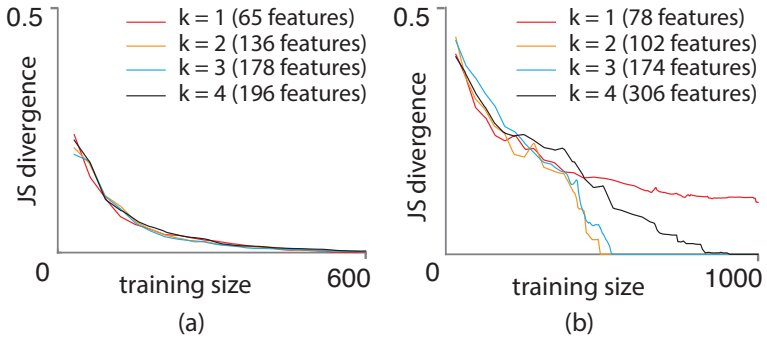


Figure 4.8 – Convergence of the pdf learning process when using sets of features associated with different maximum path lengths. We evaluate the convergence rate on the Furniture grammar (scenario F4, (a)) and the Skyscraper grammar (scenario S2, (b)). The feature sets with maximum path length 1 are shown in red, length 2 in orange, length 3 in cyan and length 4 in black.

maximum length 4 (black) slows down the convergence in comparison to our proposed scheme (maximum length 2, orange). Note that the feature set of maximum length 1 (red) is not sufficient to learn the scenario S2.

Evaluation of framework parameters. We also show an evaluation of the noise level of the kernel function (Figure 4.9a) and the weight for Lasso regularization (Figure 4.9b). We empirically select the noise level of 0.01 and a weight of 0.1 for Lasso regularization. Our results are generated based on these parameters.

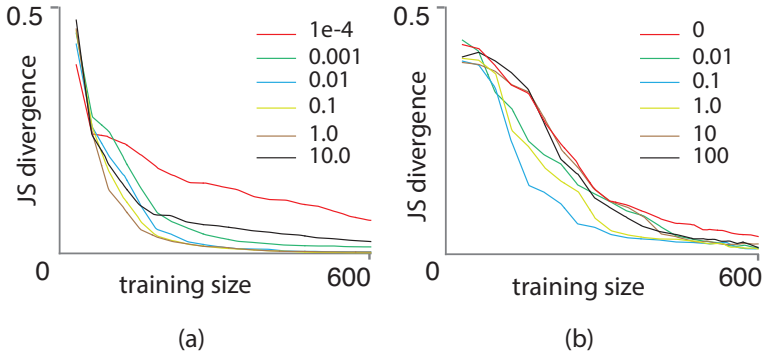


Figure 4.9 – (a) Evaluation of noise levels of the kernel function on the design scenario F1. Excessive noise (black) or limited noise (red) both result in undesirable convergence rate. (b) Evaluation of the lasso regularization factor λ on the design scenario B3. A suitable amount of lasso regularization improves the convergence in comparison to no regularization (red).

Evaluation of generation strategies. Finally, in Table 4.3, we evaluate the performance of parameter learning and structure learning by comparing the JS divergence scores of their learned pdfs with the target pdf for all design scenarios. By modifying only the grammar parameters, parameter learning cannot approximate the target pdf well (around 0.20 divergence score). Much better results can be achieved with structure learning. One exception is B3, where the design scenario involves only one rule, which is simple enough to achieve by parameter learning. Nonetheless, structure learning performs equivalently well. To separately evaluate the effect of the split operation in structure learning, we train a mixture of original grammars (without any previous splitting operations). We perform this test on the design scenario S2 and observe a JS score of 0.18096 compared to 0.0816 for a full implementation of structure learning. This low-quality output is expected as S2 requires paths of effective length 2 as features to describe the geometry of the ground block.

	F1	F2	F3	F4	S1	S2
Original	0.32824	0.44492	0.28037	0.28969	0.24646	0.31335
P-Learning	0.21976	0.25386	0.16915	0.18537	0.08854	0.19631
S-Learning	0.06850	0.06678	0.07820	0.08400	0.00090	0.08162
	B1	B2	B3	B4	A1	
Original	0.34923	0.34829	0.09858	0.11973	0.21040	
P-Learning	0.10541	0.28129	0.00710	0.12177	0.16748	
S-Learning	0.06661	0.08254	0.00712	0.07774	0.06269	

Table 4.3 – JS divergence score w.r.t the target pdf to compare parameter learning (P-Learning) and structure learning (S-Learning). The design scenarios (F1-F4, S1, S2, B1-B4, A1) are described in Table 4.2. Structure learning achieves significantly lower divergence scores. The JS scores from the original grammars (Original) are also included as a baseline for comparison.

Comparison to kernel density estimation. Figure 4.10 compares our framework to a kernel density estimation method based on Talton et al. [85] using the same number of training examples. We modify the original kernel density estimation used by Talton et al. to account for models having different preference scores by using these scores as weights. This reduces the amount of training examples needed by the original method. Still, we can observe a better convergence using our framework. In Figure 4.11 (b-d) and Figure 4.14 we compare the convergence of our method and Talton et al.'s method for different design scenarios of our grammars. In the Furniture grammar (F1), our framework needs 245 models to achieve a divergence score of 0.01 while the density estimation requires 438 models. The convergence rate is clearly better with our method

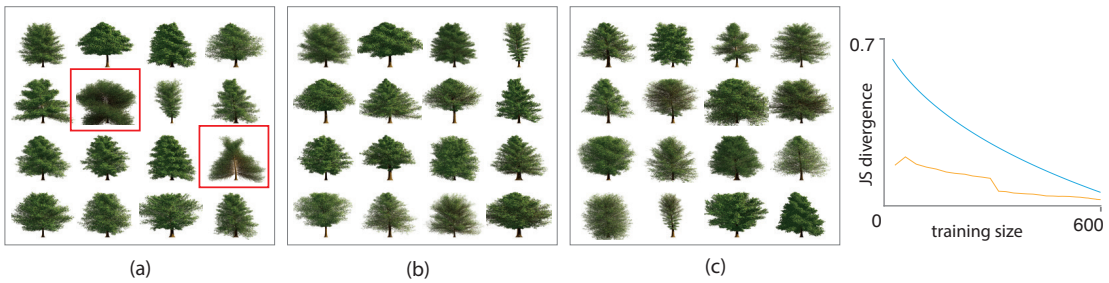


Figure 4.10 – (a) Weber & Penn trees with random parameters are often undesirable (red). Both Kernel Density Estimation [85](b) and our method (c) can bias the model distribution towards good tree samples. In verification with the ground truth using Jensen-Shannon divergence, our method (orange) converges faster than Talton et al.'s method (cyan).

in the Building grammar (B1), the Skyscraper grammar (S1), and the Airplane grammar (A1). One practical difference between our framework and Talton et al. is that we can directly give feedback about models we do not want (by giving a preference score of 0).

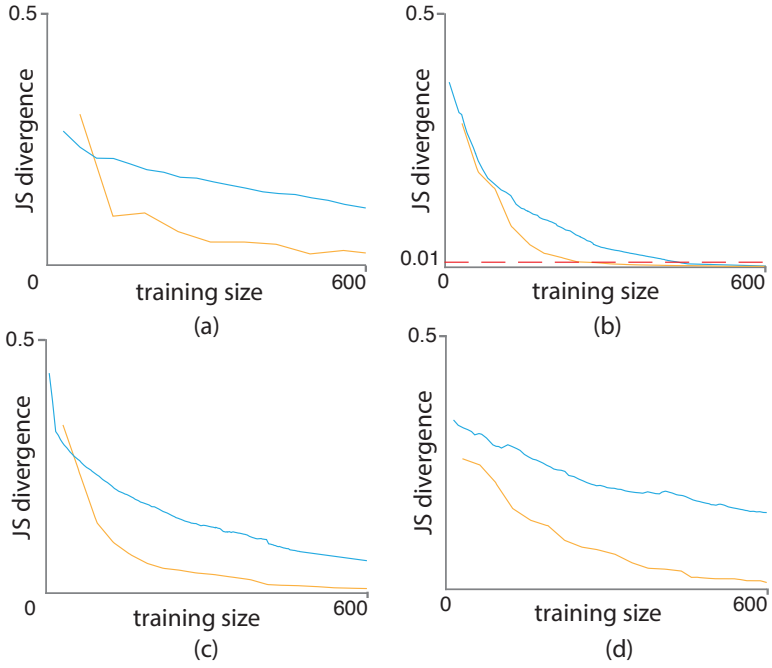


Figure 4.11 – (a) Benefit of factorizing the preference function. Training two factors F3 and F4 separately (orange) gives a better convergence than training their combination (cyan). (b - d) Our method (orange) converges faster than Talton et al.'s method (cyan) in the Furniture (F1) (b), Building (B1) (c) and Skyscraper (S1) (d) grammars.

Figure 4.12 – Using our framework, a user can design different probability density functions to bias the procedural generation towards models with desired attributes (factors). These factors can then be mixed to obtain a combined density function. We show 4×4 samples for the initial distribution (random), the factors, and the factor combinations. See Table 4.2 for the descriptions of the factors. The designed pdfs are compared with ground truth using Jensen-Shannon divergence. Color codes: F1, B1 - orange, F2, B2 - brown, F3, B3 - red, F4, B4 - green.

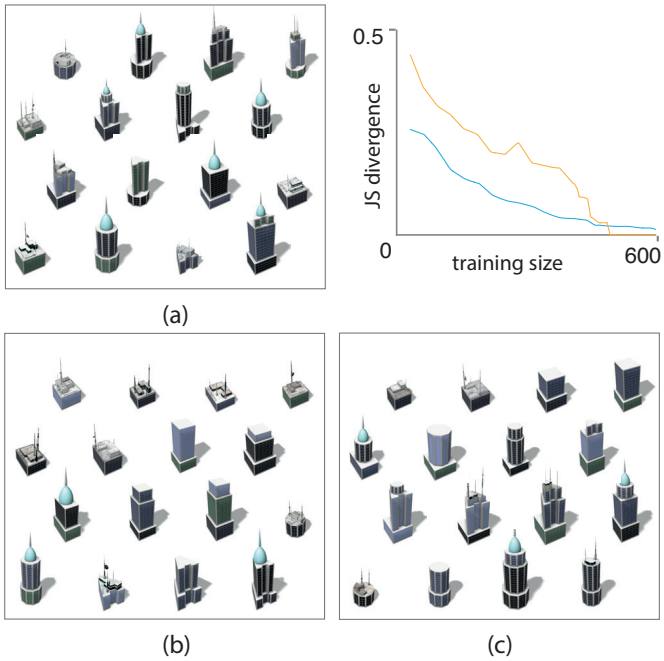


Figure 4.13 – (a) Random samples of the Skyscraper grammar. (b) Design scenario S1. (c) Design scenario S2. See Table 4.2 for the descriptions of S1 and S2. While S1 can be achieved using our initial set of features (random parameters of the shape operations, counts of the occurrence of shape labels in the derivation tree, and counts of paths, with effective length 1 in the derivation tree), S2 requires paths of length 2 as features. Jensen-Shannon divergence (S1 - blue, S2 - orange) are shown to evaluate the goodness of fit.

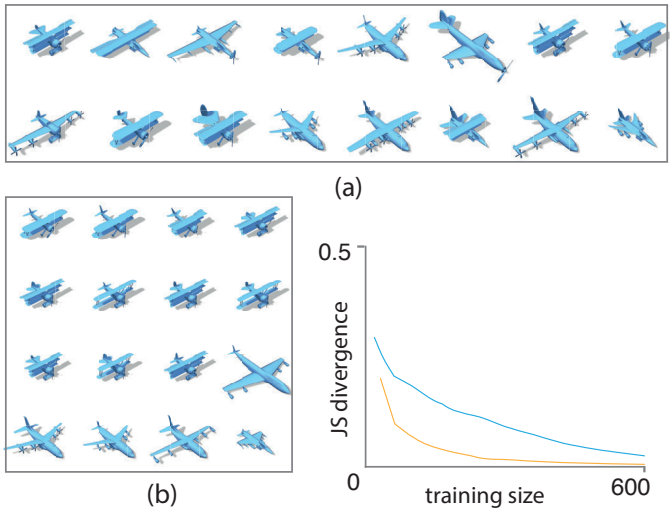


Figure 4.14 – (a) Random samples of the Airplane grammar. (b) Design scenario A1 (see Table 4.2). The JS divergence plot compares our method (orange) and Talton et al.'s method (cyan).

Figure 4.15 – We use a grammar that generates a variety of buildings for a city modeling task. (Left) A direct application of the grammar leads to undesirable results. For example, office buildings are mixed with Haussmannian buildings and small residential houses without a clear structure of different neighborhoods. We also highlight three building examples unsuitable for a city: a Haussmannian building having too many floors, upper floors with a glass facade paired with a ground floor from an apartment building, and an office building with a mansard roof. (Right) We use our framework to design three different probability density functions for this grammar, which bias towards the generation of high-rise office buildings (far), downtown Haussmannian buildings (right) and residential houses (left). We can also ensure the matching of house styles, roofs, and wall colors.

Scene Modeling. We also show a complete scene generated using our Building grammar (See Figure 4.15). We separate the city into three regions and model a different density function for each region. The parcels for placing the buildings and the streets are given.

4.8 Limitation and Future Work

4.8.1 Limitations

There are several limitations of our framework.

First, our system starts sampling models using the pdf of the original grammar. As a result, models having very low initial probability are unlikely to be sampled.

Second, in some design scenarios which involve discontinuities in the preference scores of continuous variables, our framework cannot learn the pdfs exactly with a finite number of training samples. Two examples are given in Figure 4.16. This is a typical problem in Gaussian Process Regression as one cannot represent exactly a discontinuity using a finite number of gaussians. Nonetheless, our framework can learn these scenarios when the variables are discretized.

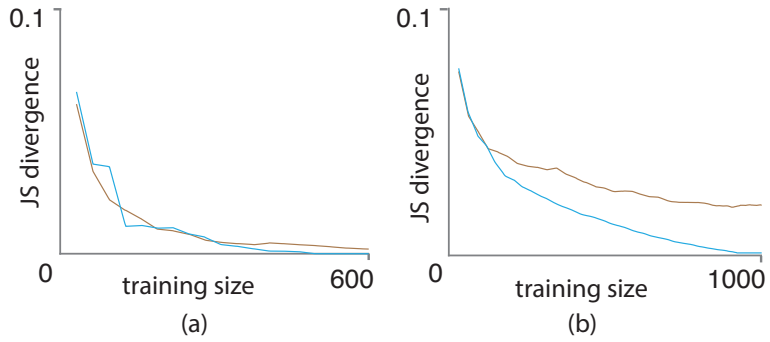


Figure 4.16 – We compare the learning accuracy of scenarios associated with continuous variables (orange) and discrete variables (with 100 discrete values each, cyan). We generate a box with three variables width, depth and height which take random values from 1.0 to 10.0. In (a), the scenario involves one variable (all boxes with height $<$ 5.0 are preferred). In (b), the scenario requires a non-linear combination of these three variables (all boxes with volume $<$ 125.0 are preferred). For continuous variables, it will require infinite number of training samples to learn the pdfs exactly. Nonetheless, our framework can learn these two scenarios with discrete variables.

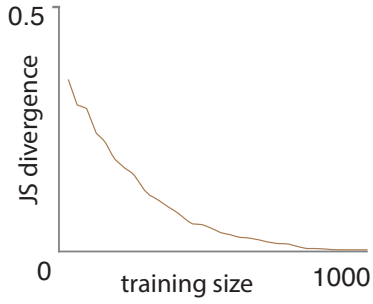


Figure 4.17 – We design an additional scenario for the Skyscraper grammar which requires the feature set with maximum path length 4. In this scenario only skyscrapers with 3 blocks are preferred and the preference score depends on the window color of the top block, in particular, skyscrapers with black windows in the top block receive a score of 50, green windows 30, blue windows 20 and other colors 0. The slow convergence is observed due to the size of the feature set.

The next limitation concerns the convergence of complex design scenarios involving features associated with long tree paths. Figure 4.17 shows a slow convergence rate associated with a new design scenario of the Skyscraper grammar which requires a large feature set (maximum path length 4).

Finally, as our features only contain branches in the production tree, our framework cannot handle complicated scenarios which require subtrees of the production tree as features. For example, in the Skyscraper grammar, one may prefer skyscrapers with a black cylinder block above a green box block but not a blue box block. As window colors and block geometries are in different branches, feature sets with only tree branches are not sufficient for this situation. Simply adding subtree features to our framework might not be a solution as this increases the number of features exponentially. This example also illustrates the fact that the success of the regression depends on how the initial grammar is written. We leave further improvement in our feature design for future research.

4.8.2 Future work

In future work we would like to extend our approach to the analysis of large shape repositories. While current approaches mainly use probabilistic models to encode shape variations (e.g. [31]), we believe that it is more useful to use grammars and probabilistic models in combination. Grammars are great to encode variations in model topology (i.e. how parts are combined) and probabilistic models are great to encode part variations

and part compatibility. Further, we would like to extend our framework to learn spatial distributions. For example, we could try to learn the distribution of residential, commercial, and industrial areas in a city or the distribution of tree species in a forest.

4.9 Concluding Remarks

In this chapter, we have presented a framework that enables a user to interactively design a probability density function for a shape grammar and to generate models according to the designed pdf. We thereby extended existing exploratory modeling tools that are suitable to select a single model from a shape space to modeling a distribution of shapes. We proposed a user interface to display, sort, and sample models to enable a user to quickly assign preference scores. To propagate user assigned preference scores to the complete procedural shape space, we proposed a novel kernel function to encode the similarity between two procedural models. This kernel function is then used for Gaussian process regression with auto-relevance detection and l_1 regularization. Finally, we introduced a structure learning method to automatically generate a new context-free grammar which approximates the learned pdf well. Our approach can benefit both non-expert and professional users to more effectively design with procedural and parametric models.

5 Discovering Structured Variations via Template Matching

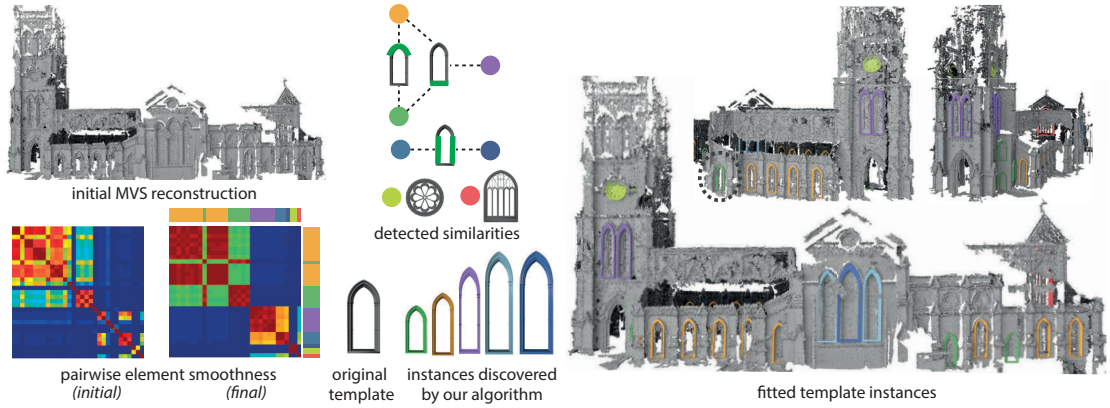


Figure 5.1 – Given a 3D acquisition of a building (e.g. MVS reconstruction) and a set of deformable templates, we present an iterative coupled template matching and deformation analysis to detect element similarities. From an initial pairwise element similarity matrix, we optimize to reveal element clusters as shown in the final similarity matrix. We show the selected templates (in green) and the similarities detected across the instances of these templates matching to each element cluster (indicated by green in the graph).

While Chapter 3 and Chapter 4 investigate algorithms to synthesize shape variations, this Chapter is dedicated to the analysis part of the geometry processing pipeline for structured variations. Understanding patterns of variation from raw measurement data remains a central goal of shape analysis. Such an understanding reveals which elements are repeated, or how elements can be derived as structured variations from a common base element. We investigate this problem in the context of 3D acquisitions of buildings. Utilizing a set of template models, we discover geometric similarities across a set of building elements.

Each template is equipped with a deformation model that defines variations of a base geometry. Central to our algorithm is a simultaneous template matching and deformation analysis that detects patterns across building elements by extracting similarities in the deformation modes of their matching templates. We demonstrate that such an analysis can successfully detect structured variations even for noisy and incomplete data.

5.1 Foreword

Many applications involving digital cities, such as mapping and navigation, heavily depend on 3D models of buildings. One way to create such content is to digitize real world scenes using different 3D acquisition technologies such as multi-view stereo reconstruction (MVS) or 3D scanning. However, many challenges arising from lighting variations, occlusions, specular surfaces still remain unsolved, and often result in noisy and partial reconstructions.

A possible approach to provide effective priors to augment reconstruction algorithms is to explore similarity patterns among building elements. However, it is extremely challenging to recover such patterns from noisy and incomplete raw acquisitions such as MVS or 3D scan data. In this chapter, we present an algorithm to discover element similarities by directly analyzing the given raw acquisitions.

We focus on detecting two types of element similarities that are common in urban scenes. Given a common base geometry and a structure-preserving deformation model, some elements are derived via identical deformation parameters and thus exhibit *full similarity*. Some elements, on the other hand, share only a subset of the deformation parameters and thus exhibit *partial similarity*. Such elements demonstrate structured variations of the base geometry, e.g. windows of Gothic style with identical arch but varying height. In order to capture this general notion of similarity, we propose to utilize a set of template models of common element types. Each template acts as a base geometry and its variations can be obtained by deforming the template. As such, we equip each template with a suitable structure-aware deformation model that defines a rich set of structured variations from the base geometry. Given such a set of templates together with the deformation model, we abstract element similarities by similarities in template deformations.

Robust template fitting is necessary to reliably detect element similarities. However,

noise and outliers in raw reconstructions unfortunately prohibit this. One way to improve template fitting is to propagate information across similar elements. However, neither the template deformations nor element similarities are known upfront, which results in a "chicken-and-egg" situation. Nonetheless, we find out that, by iterating between these two processes, the robustness of template fitting and and the accuracy of similarity detection (via analyzing the template deformations) is simultaneously improved. Hence, for each template, we analyze the template deformations to discover the subset of deformation parameters that are similar across multiple elements matching to this template. We repeat the template deformation across these elements by enforcing such parameters to stay similar.

5.2 Contributions

Our key contribution is to extract similarity patterns among a set of input elements by utilizing deformable templates. Such patterns not only reveal replicated elements, but also expose structured variations among elements that are derived from the same base geometry. Central to our analysis is the coupling of template fitting across multiple elements via the extracted deformation parameters. In contrast to prior work which focuses on individual template fitting, this coupling enables our algorithm to handle challenging datasets with significant amount of noise and missing data.

5.3 Overview

We present a template matching and deformation algorithm to analyze structured variations between building elements. Our algorithm operates on a 3D reconstruction (MVS or scan) of a building where an *element*, e.g. a window, constitutes the subset of the input reconstruction falling inside its bounding box. To facilitate the analysis, we use a set of template models of typical element types such as windows. We associate each template with a set of deformation parameters that define its structured variations (see Figure 5.4). For each of the input elements, we identify the best matching template and compute the best fitting deformation of this template, which we call a *template instance*. A key feature of the proposed algorithm is to reveal geometric similarities among the elements by detecting patterns in the deformation parameters of their matching template instances.

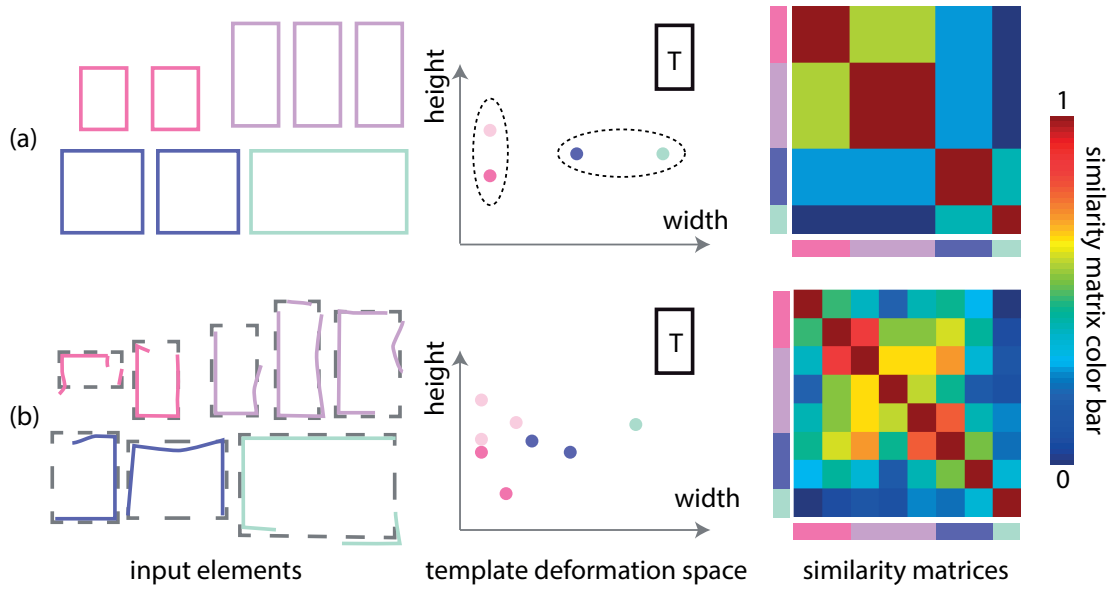


Figure 5.2 – (a) In case of perfect input data, elements that are exact replicas of the same geometry are matched with the same deformed instance of the regular template T and mapped to a single point in the associated 2D template deformation space. Loose clusters (dotted ellipses) are formed by elements with partial similarities, i.e. same width or height. (b) Clear clusters in the deformation space cannot be observed with the presence of noise and missing data. Similarity matrices computed using the pairwise element distances in the deformation space reveal this behavior. The bars on the left and bottom of the matrices identify the elements.

We motivate our proposed framework by a toy example of 2D shapes given in Figure 5.2. The scene contains 8 rectangles and our goal is to detect shapes that are exact replicas and shapes that share partial similarities i.e. same width or same height. We consider both perfect input data in Figure 5.2a and simulated noisy and partial data in Figure 5.2b. We use a rectangular template T and deform T to match the input shapes by scaling its width and height. Each deformed template instance is mapped into a 2D template deformation space, and we calculate the pairwise similarity between template instances based on the Euclidean distance in this space.

In case of perfect input data, elements which are exact replicas (rectangles with the same color) are matched to the same template instance. Elements which share partial similarities are matched to different template instances. However, deformation parameters (i.e. width or height) defining these instances are partially the same. As such, an intuitive approach to detect full and partial similarities between input elements is to deform the template independently to each element, and then detect patterns across deformation

parameters.

However, in case of real data (see Figure 5.2b), due to noise and partial data, even elements that are exact replicas are often matched to different instances of the same template. Similar elements then map to scattered points in the template deformation space which makes the aforementioned naive solution fail. To address this issue, we propose to analyze the input elements simultaneously.

Figure 5.3 illustrates main stages of our framework. We begin by deforming a set of templates to fit the given elements. We combine observations from template deformations to map each element to a common subspace representation. Intuitively, similar elements are expected to map to nearby points in this subspace resulting in small pairwise element distances. Using these distances as constraints, we consistently label each element with a deformed template instance. For each template, we discover the subset of its parameters that are similar across elements matching to different instances of the template. We repeat template deformation by coupling these parameters, i.e. enforcing them to stay similar. This iterative process consolidates information across similar elements and progressively brings them closer in the common subspace, which then reveals better clusters of elements (see Figure 5.12). We would like to emphasize that, this analysis is independent of the choice of the template deformation model.

Figure 5.3 – Given a 3D acquisition (e.g. MVS) of a building, we utilize a set of deformable templates to match its elements, i.e. windows. We combine observations from multiple template deformations via a subspace analysis to extract relations among the elements. Using these relations as constraints, we label each element with a deformed template instance (same instances are denoted in same color). We repeat template deformation by consolidating observations across elements matched to similar template instances. Performing this analysis iteratively reveals which elements are replicas of the same geometry (represented as red blocks in smoothness weight matrices) or share partial similarities (highlighted in green on the matching templates).

5.4 Simultaneous Template Matching & Deformation

Given a set of elements $\mathcal{S} := \{s_i\}$ and a set of templates $\mathcal{T} := \{T_j\}$, our goal is to detect the full and partial similarities between s_i by analyzing the template deformation. As illustrated in Figure 5.3, central to our framework is the iteration amongst four stages: 1) template deformation, 2) subspace analysis, 3) MRF labeling, and 4) similarity detection. Our analysis is independent of the chosen deformation model and a collection of the deformations models used in our experiments will be presented in Section 5.5. In the following, we briefly cover the main points of our algorithm. For more details, we refer the audience to the corresponding publication [11].

5.4.1 Template Deformation

In the first stage, we deform each template T_j to fill *all* the elements in $\mathcal{S}$ by minimizing the following energy:

$$\min_{\mathbf{d}_j} \sum_{s_i \in \mathcal{S}} E_{fit}(T_j, \mathbf{d}_j^i, s_i) + w_{sim} E_{sim}. \tag{5.1}$$

Here, $\mathbf{d}_j^i$ represents the deformation parameters of T_j^i, i.e. the instance of T_j that best fits s_i. $\mathbf{d}_j$ is a vector of deformation parameters constructed by concatenating $\mathbf{d}_j^i$ for each element s_i. The first term measures how well the template fits each element individually while the second term minimizes the difference between the deformation parameters of T_j detected as being similar across multiple elements. $E_{fit}(T_j, \mathbf{d}_j^i, s_i)$ is defined based on the chosen deformation model (see Section 5.5). As the element similarities are initially unknown, we set $E_{sim} = 0$ in the first iteration. This energy component will be updated with new similarities revealed in each iteration.

5.4.2 Subspace Analysis

Each template provides some measurement about the similarity amongst input elements by analyzing the deformation parameters of that template. The objective of this stage is to combine individual measurement from each template to extract consistent similarity information.

In particular, to capture the similarity observation form each template T_j, we associate it with a graph $G_j = (N, E_j)$ composed of a set of nodes N where each node represents

one input element, and a set of weighted edges E_j. An edge e_{ik} connecting two nodes representing input elements s_i and s_k is weighted by

$$w_{ik} = Ce^{-\|\mathbf{d}_j^i - \mathbf{d}_j^k\|^2}, \tag{5.2}$$

where C is a scaling factor.

We then combine information from all graphs G_j by adopting the subspace analysis approach of Dong et al. [21]. This method first computes a subspace representation of each graph using the corresponding graph Laplacian. Multiple subspaces are then combined into a single representative subspace U by constructing a common graph Laplacian. U maps each element to a low dimensional space and the Euclidean distance in this space will be used as the consistent pairwise distance between input elements in the subsequent step.

5.4.3 Element Labeling

Our objective in this stage is to label each s_i with the tuple $(T^i, \mathbf{d}^i)$ where T^i is the best fit template and $\mathbf{d}^i$ is the deformation parameter from T^i to s_i. This tuple defines a *matching instance* for the corresponding element. Once this labeling is established, the similarities are revealed by matching parameters in $\mathbf{d}^i$.

We first find a set $\mathcal{L}$ of potential labels. Each label $(T^i, \mathbf{d}^i)$ consists of a *discrete* template choice and *continuous* deformation parameters. One can discretize the space of deformation parameters by sparsely sampling it to obtain a discrete set of labels. However, such discretizing process does not guarantee the matching instance to geometrically fit to the element. We instead utilize the set of deformation parameters obtained in the template deformation stage to construct the potential labels. The label set is then constructed as $\mathcal{L} = \{(T_j, \mathbf{d}_j^i)\}$.

We impose smoothness assumption on the labeling process so that similar elements will be assigned similar labels. The element pairwise distance is already obtained from the previous stage. We calculate the *label pairwise distance* as the Euclidean distance between the deformation parameters if the two labels belong to the same template. For labels from different templates, the pairwise distance is assigned with a fixed value.

We formulate the labeling problem as a Markov Random Field (MRF) optimization [17]

with the label set $\mathcal{L}$ constructed above. We minimize an energy consisting of the data, the smoothness, and the label costs as follow:

$$\sum_{s_i} E_d(s_i, L_i) + w_s \sum_{s_i, s_k} E_s(s_i, s_k, L_i, L_k) + \sum_{T_j} \lambda_{T_j} E_L. \tag{5.3}$$

The data term $E_d(s_i, L_i)$ measures the fitness between the deformed template instance associated with a label to a element. It is defined as the average distance between the closest point correspondences established between the deformed template instance and the element. The smoothness term $E_s(s_i, s_k, L_i, L_k)$ enforces consistent labeling based on the element pairwise distance and the label pairwise distance described above. The last term in Equation 5.3 penalizes each unique template that appears in the final labeling. Specifically, we group the candidate labels coming from the same templates into subsets and a fixed label cost E_L (equal to $1/5^{th}$ of the average data cost in our experiments) is induced if at least one label is used from such a subset. The indicator variable λ_{T_j} is set to 1 if an instance of the template T_j appears in the final labeling.

Due to noise and partial data, replicated elements initially often map to different labels. The smoothness term progressively enforces these elements to be assigned to the same label and thus brings them closer. This leads to the formation of red blocks in the pairwise smoothness matrices in the final iterations of our algorithm (see Figure 5.9). If there are similar templates, similar elements might get assigned to instances of different templates. The label cost favors the use of as few unique templates as possible and thus enables a consistent labeling. Both smoothness and label costs enforce the selection of fewer templates. Hence, elements that exhibit variations of a base geometry are matched to different instances of the same template (see Table 5.1).

5.4.4 Similarity Detection

Once each element is labeled with a matching template instance, we evaluate the labels to extract a set of similarity relations $R = \{r\}$ among the elements. In particular, for each pair of elements s_i and s_k which are labeled with two labels $(T^j, \mathbf{d}^i)$ and $(T^j, \mathbf{d}^k)$ belonging to the sample template, we define the relation $r = (s_i, s_k, \mathbf{c}_j, T_j)$. The vector $\mathbf{c}_j$ is a binary vector of size equal to the number of deformation parameters of T_j which encodes the matching between entries of $\mathbf{d}^i$ and $\mathbf{d}^k$. In specific, if the difference between two corresponding entries of $\mathbf{d}^i$ and $\mathbf{d}^k$ is smaller than a pre-defined threshold, we set the corresponding entry of $\mathbf{c}_j$ to one. The threshold is set to 0.5% of the maximum

diagonal length of the element bounding boxes. Such *coupled parameters* indicate partial similarities between s_i and s_k with respect to template T_j. A vector $\mathbf{c}_j$ consisting of all ones ($\mathbf{c}_j = \mathbf{1}$) indicates that s_i and s_k are exact replicas. In this case, we define an additional relation for these elements for all other templates with $\mathbf{c}_j = \mathbf{1}$.

In subsequent iterations of our algorithm, we enforce the coupled parameters to stay similar during the template deformation. To do so, we update the E_{sim} term in Equation (5.1) as follow:

$$E_{sim} = \sum_{(r)\in R} E_{dist}(r). \tag{5.4}$$

Here, the term $E_{dist}(r) = \mathbf{c}_j^T(\mathbf{d}_j^i - \mathbf{d}_j^k)^2$ for $r = (s_i, s_k, \mathbf{c}_j, T_j)$ penaltizes the difference between the coupled deformation parameters of the new instances of T_j that fit s_i and s_k. With the new deformation parameters, we repeat the labeling process as described above. The iterative process terminates when there is no change in the final labeling.

5.4.5 Extension to Large Template Sets

When performing the proposed coupled analysis, it is possible to consider every template-element pair for deformation, especially if the number of utilized templates is low. However, as the size of the template set grows, a pre-organization of the templates becomes necessary. This organization not only reduces computational complexity but also provides high-level relations among the templates that can be used to guide the template matching process.

In our experiments, we propose a simple strategy to group the templates based on their deformation capabilities (see Section 5.6). Given a grouping of the templates, we first identify the best fitting group for each element by aligning the bounding boxes of each template-element pair via a similarity transformation and compute the fitting error. We select the template group with the minimum average of such fitting errors. For each element, we deform templates only in its matching group. Each group of templates are thus deformed only to a subset of the elements and we perform the subspace analysis independently for each such subset. For every pair of elements mapped to the same subspace, we compute a pairwise smoothness weight as explained before. Any pair of elements mapped to different subspaces are assigned a smoothness weight of 0. During labeling optimization, we construct a common candidate label set for all the elements

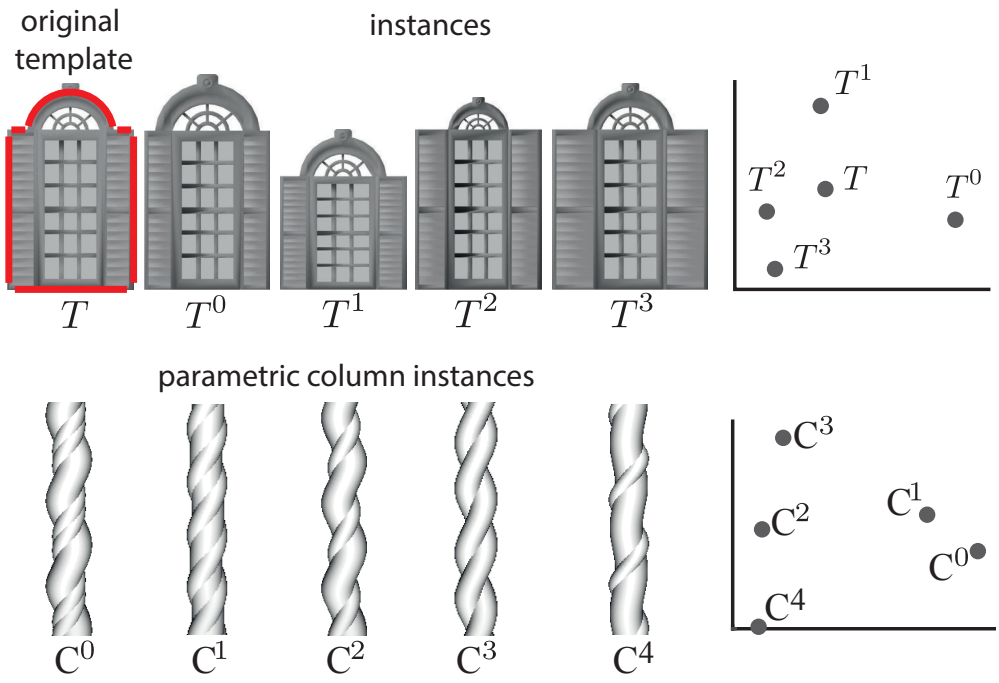


Figure 5.4 – For the template T equipped with the i-Wires deformation model, we illustrate various instances $(T^0, ..., T^3)$ with different parameters of the detected feature wires (shown in red). We also show several column instances generated by a parametric model. Each instance is visualized as a point in the corresponding deformation space using multi-dimensional scaling projection.

irrespective of their matching template groups. This enables an element to be possibly matched to a template instance from a different group and thus recover from any error that might have occurred during the initial group matching.

5.5 Template-Based Deformation

Given a deformation model, we have presented a template matching algorithm. We evaluate this algorithm by adopting two deformation models suitable for architectural data. For element types with dominant feature lines such as windows or doors we adopt the structure-aware i-Wires deformation model [27]. We also demonstrate an example of a parametric model on curved columns. Each deformation model defines the space of structured variations for each base template. In Figure 5.4 we visualize structured variations associated with a window using the i-Wires model and variations of a helical column using the parametric column model. Some non-trivial structured variations such as T^2 and C^4 can be obtained by using this generalized concept of structure. We next

provide a brief description of these deformation models.

i-Wires deformation model. In an offline pre-processing stage, we equip each template with a set of *feature wires* extracted by the algorithm presented in [27]. These feature wires are associated with "sharp" edges in the template mesh. They are compound wires consisting of one or more parametric *atomic subwires*. In our system, we consider the two types of atomic subwire: 1) straight line and 2) circular arc. We parametrize a straight line by its length and direction and a circular arc by the center, the radius and the arc opening angle. A wire is then parametrized by the combination of parameters of its subwires. We also identify in this stage structural information of the template, such as reflection symmetry or planarity of feature wires. All the structural information is encoded as constraints between subwire parameters and will be preserved during the deformation.

During the template deformation process, we optimize for the wire parameter to fit the template wire features to the 3D line features of the point cloud. The 3D line features are calculated by applying multiview stereo matching on 2D edges of input images [2].

Parametric deformation model. We use a parametric deformation model to demonstrate our analysis on element types such as curved columns. Our parametric model is based on a *helical structure*, i.e. a circle swept along a 3D helix curve. A variety of columns can be generated by applying CSG operations, i.e. union or difference, to a set of helixes. Each helical structure is characterized by the *pitch*, *radius*, and *start angle* of its helix and the *radius* of its swept circle.

The parametric model we use to generate curved columns acts an abstract template where each parametric column is an instance of this template. We begin our analysis by first generating a set of concrete column instances given the input elements. We cut each input element along its up direction and fit circles to the resulting outline curves. We generate helical structures in a RANSAC-like fashion by finding groups of circles that are related by the same pitch and thus are possibly swept along the same helix. We then evaluate different combinations of such helicals, i.e. different number and different CSG operations, to generate candidate columns fitting the element. Starting from this initial set of parametric columns, our coupled analysis identifies the best fitting column instance for each element. Column parameters are further optimized by coupling the parameters of the individual helical structures of the column instances detected as similar.

5.6 Evaluation

Datasets. We demonstrate our algorithm both on MVS data (we compute camera parameters using the VisualSFM tool [92] and the dense reconstruction by PMVS [26]) and scans acquired with Microsoft Kinect (using the software Skanect [1]).

Element selection. Our algorithm detects similarity patterns among a set of building elements. Even though there exist automatic facade parsing methods exploiting the presence of horizontal and vertical splitting lines and regular grids [60], we observe that such methods fail to identify the elements of more complex architectural scenes. Instead, we utilize a semi-automatic approach. For MVS reconstructions, we adopt the method of Ceylan et al. [10] which requires the user to roughly mark an element of interest, e.g a window frame, in only *one* of the input images and automatically detects its repetitions. We revert to user input to mark any missing element in case of strong variation or large occlusions. For scan data, we require the user to mark the elements in 3D by defining their bounding boxes. Table 5.1 provides the number of user marked elements for each of our examples. Note that although additional user input to cluster similar elements may improve the results, this is not sufficient to detect partial similarities across the clusters. Such similarities are difficult to manually specify as they may not be obvious by visual inspection and require the user to mentally solve the template selection and deformation problems simultaneously. Therefore, once elements are identified, we revert to our automatic analysis with no use of prior information to detect both full and partial element similarities.

Template set. Our template set consists of 60 models downloaded from the Digimation Model Bank and Trimble 3D Warehouse (Figure 5.5). For computational efficiency, we pre-organize the database into groups. We deform each pair of templates to fit the other and define a pairwise distance equal to the maximum of the resulting fitting errors. We then cluster the templates based on these pairwise distances. This simple strategy successfully distinguishes between circular, arch, and rectangular windows but leads to confusion between arch and triangle-top windows. We revert to user input to regroup the few misclassified templates. We note that it is possible to propose a different organization of the templates, e.g based on architecture style.

We also evaluate our algorithm using a parametric model that can generate helical columns.

[1] http://skanect.occipital.com

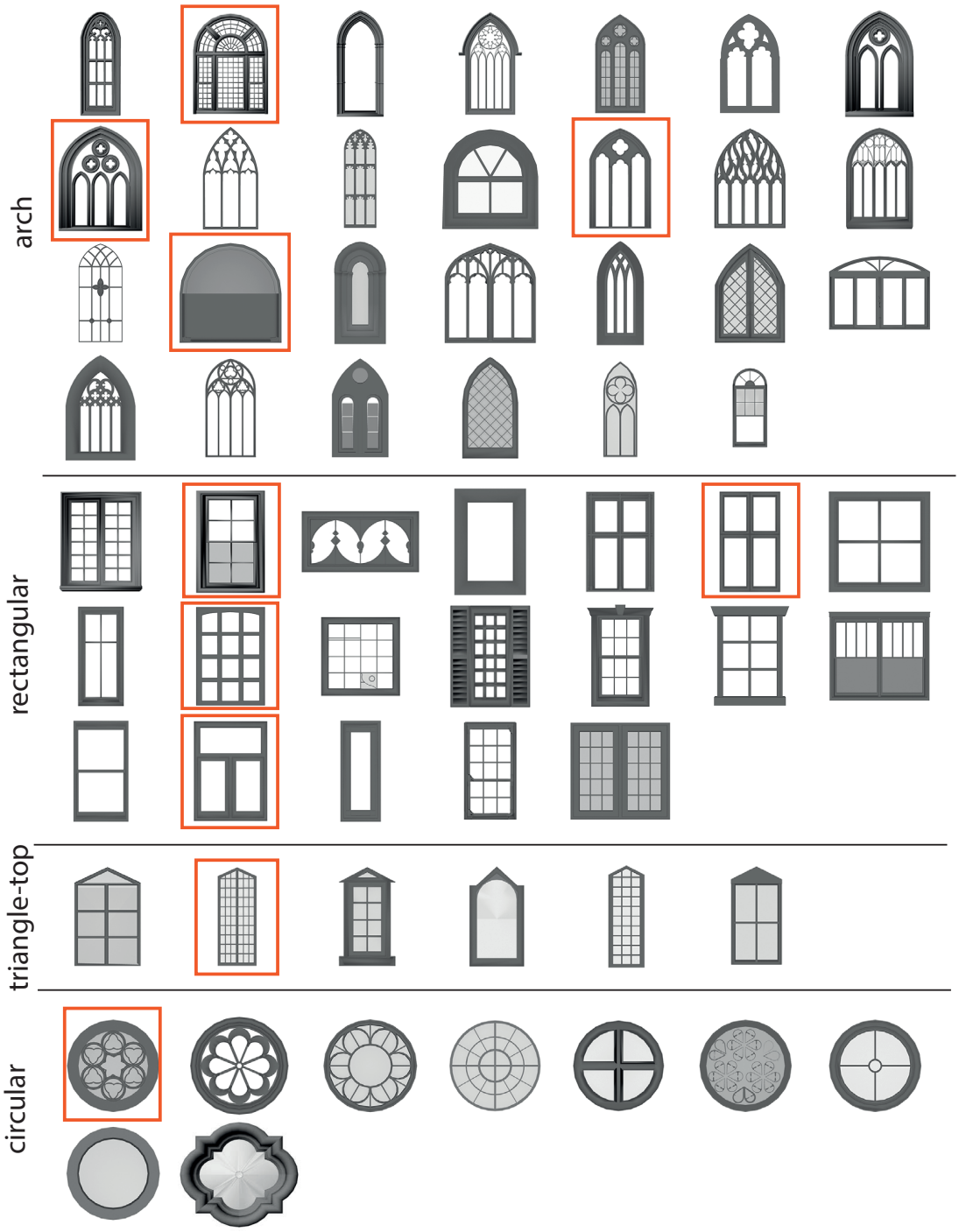


Figure 5.5 – Template database. Templates used for evaluation in Fig 5.7b are shown in red.

5.6.1 Performance on Synthetic Data

We evaluate our algorithm by changing the amount of variation across input elements, the number of utilized templates, and the data quality. In order to assess each factor independently, we perform evaluations on synthetic data using a fixed set of parameters.

Effect of element deformations. Our algorithm labels each input element with a matching template instance by incorporating data and smoothness terms. While the data term evaluates the individual label assignments, the smoothness term favors similar labels for similar elements. Due to this coupling, the amount of variation among the elements affects the final choice of labels. We illustrate this effect on a set of synthetic elements created by gradually increasing the variation among them (see Figure 5.6). When the template set includes a template capable of capturing all of these variations,

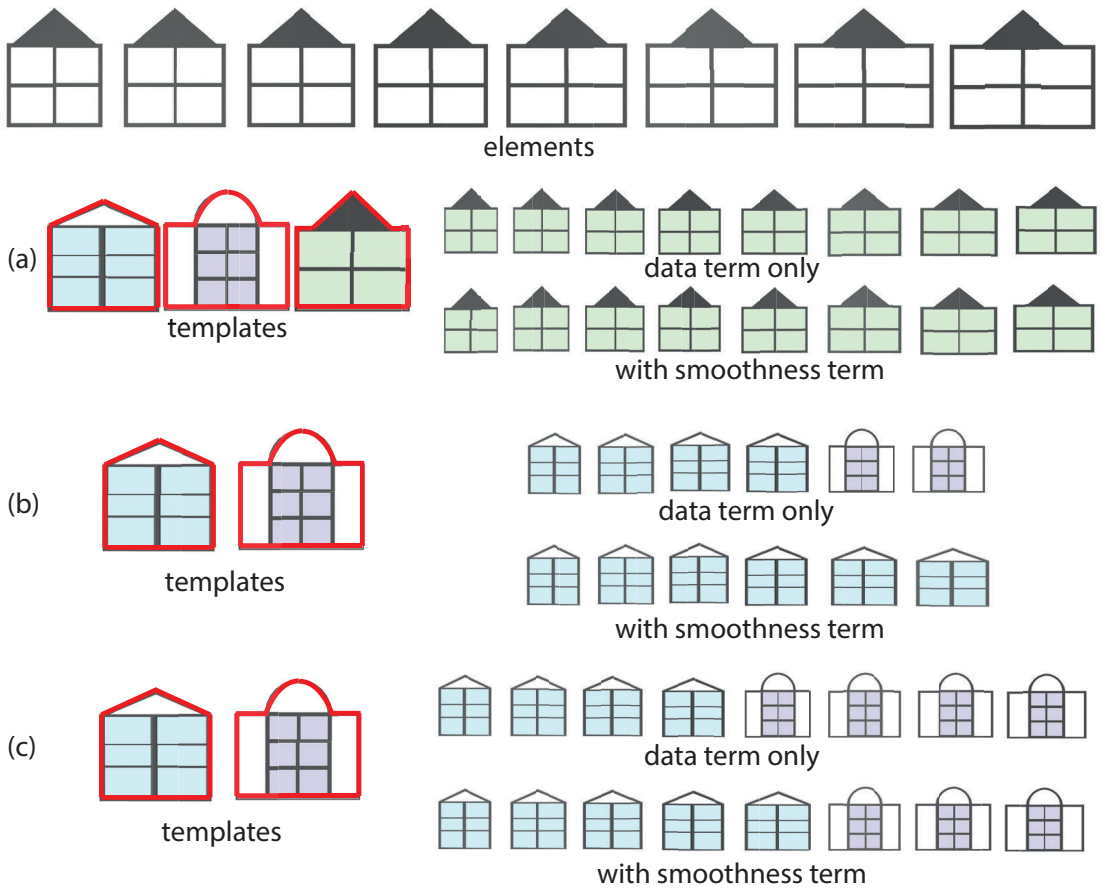


Figure 5.6 – The amount of variation among the elements affects the final choice of template instances. For different sets of templates (with feature wires shown in red) and elements, we show the selected template instances based on data term only and additionally considering the smoothness term.

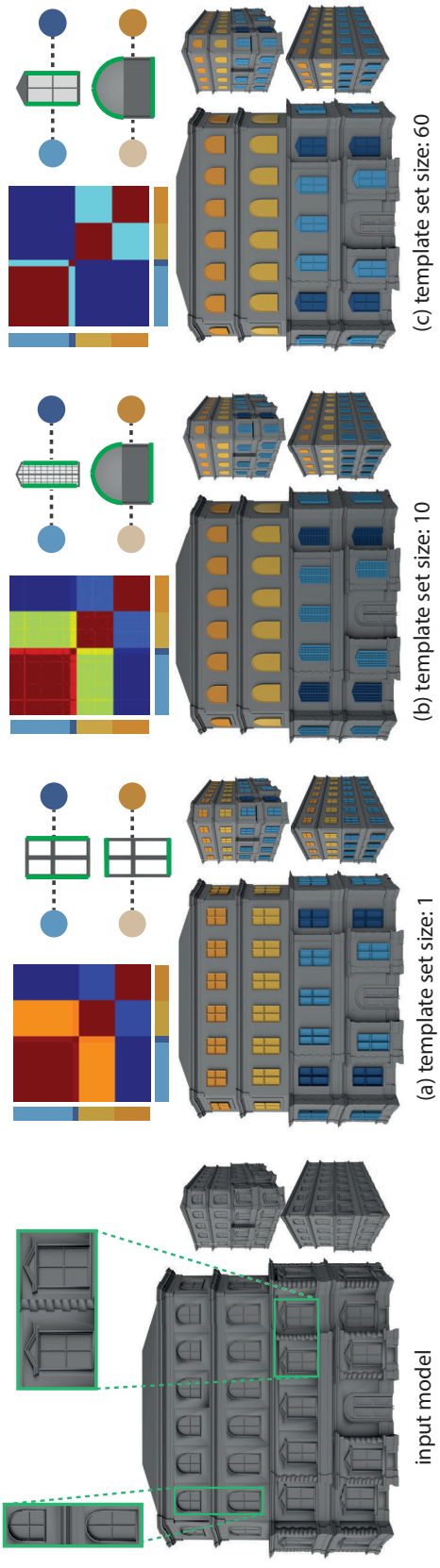


Figure 5.7 – We show the selected template instances for a synthetic house model (consisting of 38 narrow triangle-top, 4 wide triangle-top, 23 long arch, and 23 short arch windows) when using different number of templates. For each case, we also show the color-coded smoothness matrices and the partial similarities detected between the elements (highlighted in green). Note that the removal of the triangle-top template selected in (c) results in a selection of another triangle-top window in (b).

all elements are labeled with different instances of this template (Figure 5.6a), i.e. the smoothness term has no effect. When we remove this template, however, none of the remaining templates is capable of perfectly capturing the element variations. We first consider the first six elements, where four of them prefer the first template based on the data term only. Even though the fifth and sixth elements prefer the second template based on the data term, the smoothness term enforces them to pick labels from the first template (Figure 5.6b). We then add two more elements that also prefer the second template individually. This is perceived as a strong indication that the second template is also a likely assignment. Thus the last three elements are now assigned to labels from the second template (Figure 5.6c).

Effect of number of templates. Since templates deform similarly to fit similar elements, each template contributes to detection of element similarities. We illustrate this on a synthetic house model with two types of windows showing variation in height and width (see Figure 5.7). We run our algorithm using an increasing template set size of 1, 10 (selected templates are highlighted in Figure 5.5), and 60 (organized as four groups). In each case, we show the pairwise element smoothness weights in color-coded matrices. For each block of identical elements, the side color bars denote the color of the corresponding matching template instance. We show the partial similarities detected between such instances in a graph by highlighting the coupled parts of templates in green.

We observe that even a single template is capable of distinguishing the variation among the elements resulting in the selection of four distinct template instances (Figure 5.7a). With additional templates, the two window types are identified resulting in the selection of two instances of each template (Figure 5.7b). When using a grouping among the templates (Figure 5.7c), the two type of window elements are initially matched to different template groups resulting in no smoothness relation between them, i.e. blue blocks in the corresponding smoothness matrix. With a single rectangular template, however, the similarity between the height of the triangle-top and long arch windows is reflected as a reasonably high smoothness weight, i.e. orange block in the corresponding matrix.

Effect of data quality. The input data quality has a direct impact on template deformations. We compare the performance of our algorithm on synthetic data (Figure 5.7c) and a MVS reconstruction obtained from the rendered images of the model (Figure 5.8). We observe two main sources of error that potentially influence our results. First, due to the challenges in correspondence search or limited sensor resolution, 3D reconstructions

exhibit a *general* degradation in data quality which might lead to failure in capturing fine details. Even though our algorithm selects the same templates as in the ground truth case, it fails to capture the subtle variation in the width of the two instances of triangle-top windows (Figure 5.8a). Second, factors such as large occlusions result in *local* degradation in data quality. Thus, when we place a large tree model in front of the house, we cannot recover the correct template assignments for the windows occluded by this tree (Figure 5.8b).

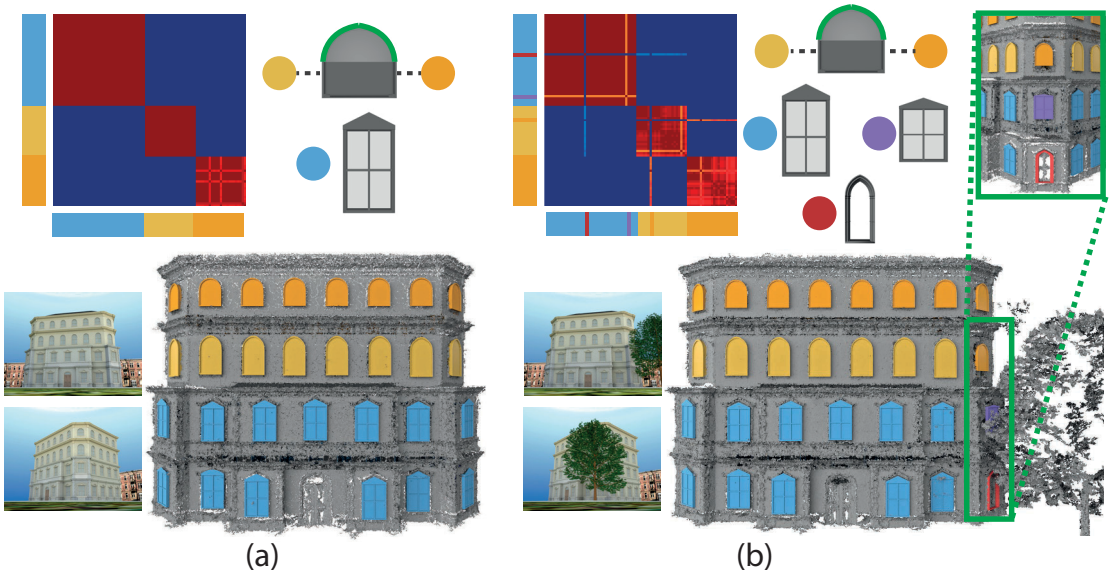


Figure 5.8 – We evaluate our algorithm on MVS reconstructions obtained from rendered images of a synthetic model. Due to loss of fine details, we cannot recover the subtle variation in width of the triangle-top windows in blue (a) and the occlusion by a large tree results in wrong template assignments for some elements (b).

5.6.2 Performance on Real Data

We evaluate our algorithm on various real datasets with different style and varying complexity (Figure 5.1, 5.3, 5.9, and 5.10). Table 5.1 shows the statistics of our algorithm on each dataset.

Performance on MVS output. In our evaluations, we mainly focus on challenging MVS output that suffers from significant amount of noise and missing data.

Our algorithm successfully discovers both full and partial similarities between elements. Due to noise and missing data, such similarities are difficult to capture with traditional

	N_i	N_s	N_e	T_d	T_c	T_i
Dataset 1	120	7	32	8	3	7
Dataset 2	60	3	39	10	2	3
Dataset 3	160	8	32	12	4	10
Dataset 4	299	10	99	22	7	9
Dataset 5	70	13	25	7	5	9
Dataset 6	129	6	57	13	3	4
Dataset 7	126	7	17	7	4	8
Dataset 8	-	36	36	-	-	10

Table 5.1 – The table shows the number of input images (N_i), the number of user selected elements (N_s), the total number of detected elements (N_e), the numbers of templates selected by the independent analysis (T_d) and with the coupled analysis (T_c), and the total number of template instances discovered (T_i). Note that for Dataset 8 we use a parametric model considered as a single template.

methods. In particular, in *Dataset 1*, 30 windows out of 32 building elements in total are assigned as structured variations of the same template (see Figure 5.1) whereas template fitting for each element individually results in the selection of 5 different templates (Figure 5.12). The detected structured variations for other MVS datasets are shown in Figure 5.9.

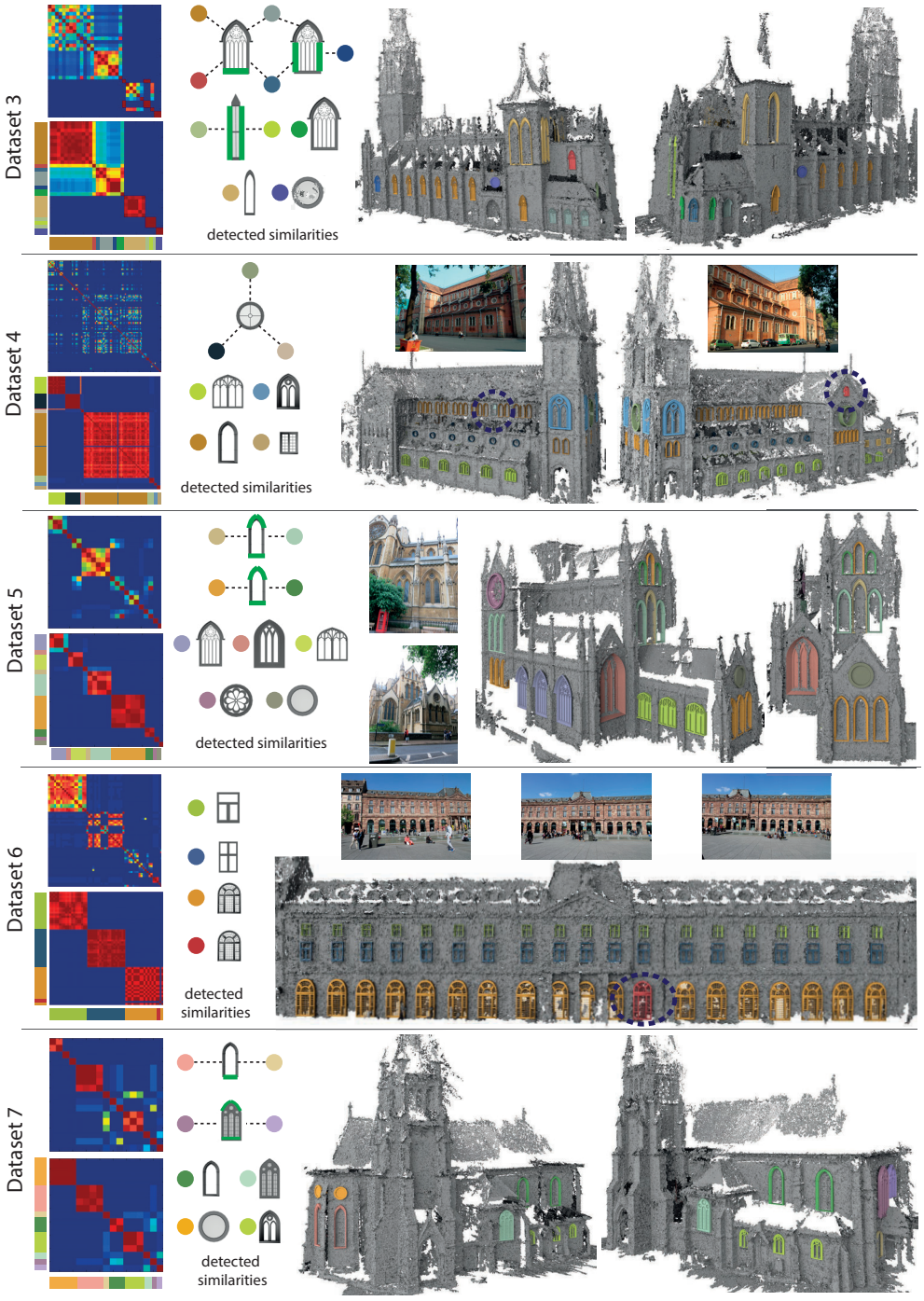


Figure 5.9 – For each data set, we show the smoothness matrices in the first (top left) and final (bottom left) iterations of our algorithm. Color bars at the sides of the matrices denote the color of the matching template instances of the corresponding block of identical elements. Partial similarities detected between different element blocks are shown on the corresponding templates in green. We denote the elements matched to wrong template instances with dotted circles. Please refer to Table 5.1 for details.

Figure 5.10 – We use a parametric deformation model to match the helical columns in a 3D scan of a museum. We show the smoothness matrices in the first (top-left) and final (bottom-left) iterations of our algorithm. We omit the elements in the bottom floor, which have been detected as identical, from these matrices for visualization purposes. The side color bars denote the color of the matching column instance of the corresponding block of identical elements. Each column instance is composed of a number of individual helical structures that we show in gray (e.g. the red instance is composed of 4 helical structures). We show the similarities detected across these substructures in solid edges: identical (blue), reflected (orange), same pitch only (purple).

Performance on scan data. We also evaluate our algorithm on a scan of an indoor scene containing curved columns (see Figure 5.10) using a parametric deformation model. This deformation model captures the properties of the individual helical structures each column is composed of. Starting from a candidate set of columns fitted individually to each element, we discover similarities across these individual helical structures by our simultaneous template deformation framework: identical, reflected (i.e. with opposite rotation direction), and sharing the same pitch angle only. We also apply our algorithm to analyze the window elements in the MVS reconstruction of the same scene (obtained from 400 images) and detect that they are identical.

To evaluate the robustness of our algorithm, we have synthetically added noise to this dataset and re-performed our analysis (see Figure 5.11). Even though some of the fine details are not captured due to noise, our analysis is stable and recovers the expected similarity patterns.

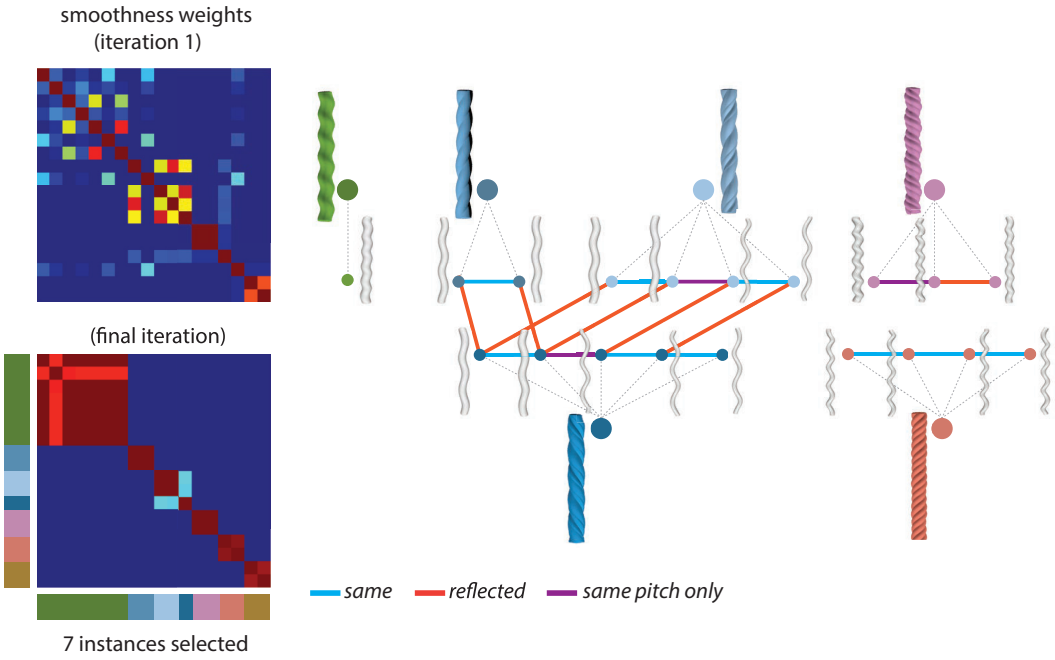


Figure 5.11 – We synthetically add noise to the input scan by uniformly disturbing each vertex in the range $[-4d, 4d]$ where d is the average local sample spacing. The smoothness matrices computed in the first and final iterations of our algorithm for the same set of elements as in Figure 5.10 are shown together with the detected similarities (solid color edges).

Comparison to naive clustering. Due to noise and partial data, an independent analysis of each element often results in the selection of different templates for elements

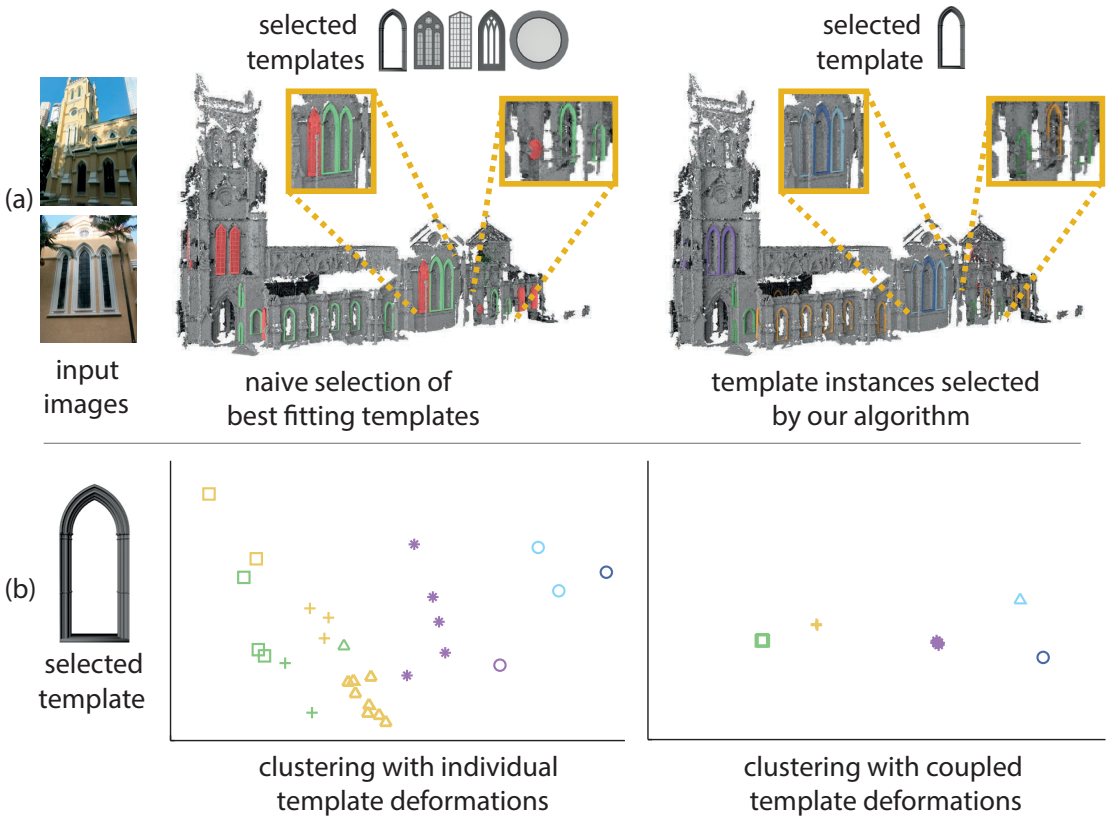


Figure 5.12 – (*a*) Individual template fitting for a set of elements results in the selection of 5 different templates whereas our algorithm assigns the elements to 5 different instances of the same template. (*b*) Given a template selection, we visualize each element in the low-dimensional deformation space of the template (replicated elements are shown in same color) using the deformation parameters obtained by individual fitting vs. our algorithm. The clusters generated by the k-means ($k = 5$) algorithm are indicated by different symbols. Note how clusters on the left span different template instances and lead to misclassification.

that are derived from the same base geometry (Figure 5.12a, left). Even if we annotate the correct template selection, replicated elements are mapped to scattered points in the template deformation space. Thus, standard clustering algorithms such as *k-means* fail to identify the correct element clusters (Figure 5.12b, left). Our iterative analysis, however, progressively consolidates observations across similar elements and reveals distinct element clusters (Figure 5.12b, right). For this dataset, compared to ground truth clustering by visual inspection, the clustering of our algorithm achieves a mutual information score [90] of 0.876 with one mislabeled element whereas the clustering based on independent analysis has a score of 0.468.

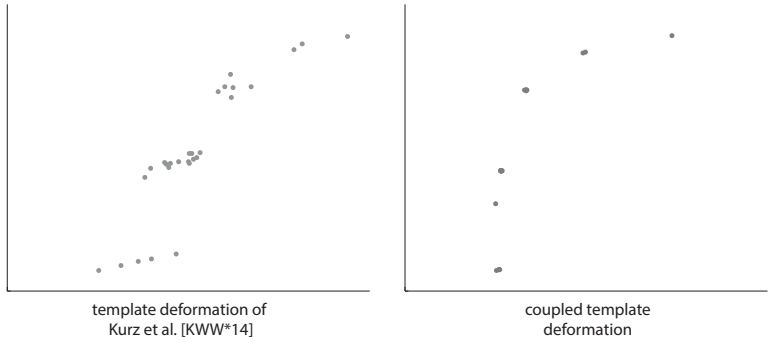


Figure 5.13 – Given the correct template selection, the template deformation model of Kurz et al. [38] maps replicated elements to scattered points as a result of individual fitting. Our simultaneous analysis, on the other hand, forms tight clusters. The method of Kurz et al. provides a free-form deformation. Thus, we parameterize the deformed templates by the width and height of their bounding boxes.

Comparison to Kurz et al. [38]. For the dataset shown in Figure 5.12, we also evaluate the method of Kurz et al. [38]. This method presents a template deformation model that explicitly considers the symmetric features of the templates. Even with such an advanced deformation model, independent template fitting for each element maps the replicated elements to scattered points (Figure 5.13, left) whereas our analysis successfully produces tight clusters (Figure 5.13, right). Note that the method of Kurz et al. is complementary to our analysis since it can be used as the input template deformation model.

Effect of parameters. Our analysis involves a small set of parameters that are listed in Table 5.2. For most of these parameters we use the default values introduced in Section 5.4 in all of our evaluations. The only parameter that requires adjustment is w_s, the weight of the smoothness term involved in the MRF optimization. An inherent challenge in analyzing raw data measurements as obtained from MVS or scanning is to distinguish noise from fine details. Our algorithm solves a labeling problem consisting of data and smoothness terms to reflect this tradeoff. A lower w_s helps to capture high frequency details in case of reliable data. In case of noisy and partial data, e.g., MVS

	MVS data	scan data
w_{sim} (Eq. 5.1)	10	10
E_L (Eq. 5.3)	$1/5^{th}$ of avg. data cost	$1/5^{th}$ of avg. data cost
w_s (Eq. 5.3)	0.1	0

Table 5.2 – The table shows the parameters involved in our analysis and their values used in our evaluations.

data, however, the data term becomes unreliable and a higher w_s allows consolidating observations across multiple elements. In our evaluations, we set $w_s = 0.1$ for all the MVS examples. We disable the smoothness term for the scan data since the data quality is reliable and there are subtle variations across the elements we would like to capture (e.g. the dark and light green columns in Figure 5.10).

Performance evaluation. We run our experiments on a 2.8 GHz Intel Core i7 machine. Our analysis is iterative where each iteration begins with the *template deformation* stage. We group the template models based on their deformation capabilities and identify the best matching group of each input element. For each element, we deform the templates only in its matching group. Given n input elements and k templates in each template group in average (15 in our evaluations), we perform $O(nk)$ template deformations. We then map the observations from the deformations to a common space using the *subspace analysis*. This step is almost instantaneous and has negligible time complexity (700 milliseconds in average). We then perform the *labeling optimization* to label each input element with a deformed template instance. We choose the best 3 fitting template instances for each element to construct a label set of $3n$ labels. Given n elements and $3n$ labels, this step takes 80 seconds in average. Finally, the *similarity detection* step extracts the coupled template parameters across the selected template instances. This step also has negligible time complexity (80 milliseconds in average). For all of our datasets, our analysis converges in $5 - 6$ iterations. The computational complexity is dominated by the template deformation step where the time spent for each deformation depends on the choice of the deformation model.

Limitations. Due to limited sensor resolution, 3D reconstructions often fail to capture fine details, e.g. in the substructures of the elements. Such missing details or the lack of a more suitable template might result in selection of a template different than a user-intended one. For *Dataset 4*, the closest template has been selected to capture the two-arch structure of the windows shown in green (see Figure 5.9). For *Dataset 8*, we have failed to capture the subtle details in the column shown by the blue rectangle and our parametric model is not capable of generating the details on the column shown by the green rectangle (see Figure 5.10).

Severe local degradations in data quality, e.g. due to large occlusions, prevent reliable template deformation and is another source of failure for our algorithm. In our examples, we indicate such failures by dotted ellipses (Figure 5.1 and 5.9). In Figure 5.14, we demonstrate two challenging cases, where almost half of the indicated windows are

occluded. Even though our algorithm identifies the correct template, it discovers the wrong (shorter) instance.

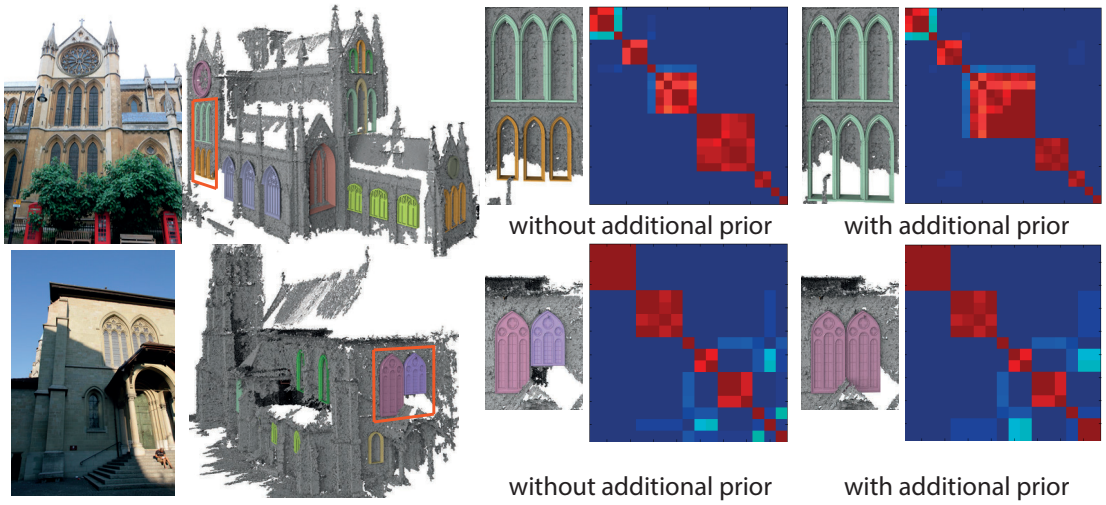


Figure 5.14 – Our algorithm fails to match the windows indicated in orange to the correct template instances due to large occlusions. It is possible to augment our analysis with additional priors, e.g. incorporating smoothness constraints among elements arranged in a grid, to resolve such failures. We show the element smoothness matrices with and without use of such priors.

5.7 Concluding Remarks

We presented an algorithm to discover similarity patterns among a set of elements by using deformable template models. Our template matching and deformation analysis identifies the best fitting template instance of each element and detects patterns in the deformation modes of these instances. Even though it is possible to incorporate additional priors, we assumed no additional information to demonstrate the effectiveness of the approach. Our approach is independent of the choice of the deformation model. Thus it can be adopted to other problem settings by defining other context-specific templates.

In our evaluations, we utilized a simple template grouping strategy. Considering a more sophisticated organization, e.g. a hierarchical grouping, is an interesting future direction. Exploring additional priors, e.g. style-annotated templates, both in template organization and deformation can aid tasks such as discovering similarities across different buildings.

Accompanying our algorithm with a deformation model that supports discrete parameters

will enable to capture discrete changes among elements, e.g. in their substructures.

We have adopted a semi-automatic approach to identify the elements of a building. Learning common element deformations to enable automatic detection of building elements is an interesting future direction.

6 Conclusions

This thesis has presented algorithms to process structured variations for three common types of data in computer graphics including 1) building facade textures, 2) procedural modeling production, and 3) 3D reconstruction point clouds.

High quality facade textures are needed in various applications especially in the domain of urban reconstruction. We designed algorithms to process structured variations in building facades even with irregular structures. We used decomposition trees to represent the spatial relations between facade elements. We additionally introduced generalized grids to encode structural relations of the input facades. Different from earlier attempts to capture structural information in terms of geometric symmetries and regularities, generalized grids provide additional potential links between non-symmetric elements with non-regular element spacing. We proposed specialized discrete and continuous operations to modify facades while preserving the key spatial and structural relations. While ambiguity is a core challenge in structure-aware editing pipelines, by means of an iterative editing process, our framework gives access to a large design space while avoiding exposure to an exponential set of alternative solutions.

We have shown that, it is possible to control the likelihood of a procedural model being generated in a procedural modeling system by interactively designing probability density functions for shape grammars. The system learns the pdf from users via active learning. By means of the novel structure learning algorithm, our system automatically generates a new context-free grammar to approximate the learned pdf. The proposed system enables novel applications for both expert and casual procedural modelers, for example to explore the space of procedural models, to fine tune the distributions of generated models or to

effectively remove unwanted models out of the space of generated models. We believe this system provides the basics for future research, especially in the domain of using procedural modeling to generate high quality and customized structured variations.

While the above two frameworks focus more on the synthesis of structure variations, we demonstrated an analysis framework to detect full and partial similarities between semantic elements of 3D reconstruction output. We proposed a new formulation of shape structure, which is the matching of deformation parameters from common base geometries. By using templates as the common base geometries, and coupling between template matching and deformation parameter analysis, the proposed system efficiently reveals useful patterns of structured variations even from noisy point clouds with missing data. Such analysis results can be used to improve the reconstruction results by means of 3D in-painting algorithms and structure-aware editing or provide suggestions for artists to design similar models.

The central idea of this thesis work is that, instead of processing individual object separately, analyzing and modeling structured variations is a good way to accelerate the design process, especially in large scale projects such as modeling digital cities. The ultimate goal of this research, therefore, is to inspire future research in reducing the complexity in digital content creation by means of utilizing the concept of shape structure and processing structured variations. There are certain advantages in combining the analysis and synthesis step of structured variations. For instance, once detecting that elements of building facade such as windows and doors are often arranged in a grid, we can enforced a grid structure when synthesizing new facades. On the other hand, understanding how new shapes are created, e.g. a facade can be resized by adding new floors, we can update semantic priors, e.g. horizontal alignment in the analysis step. While such combination is not yet demonstrated in this thesis, we believe the link between analysis and synthesis can be a key methodological element which facilitates the processing of structured variations in particular and content creation in general. Our thesis work hopefully set the foundations for such integration in future research.

A Furniture and Building Styles

In this appendix, we visualize the styles for furniture and buildings defined in Table 4.2.

1. Consistency of table legs and leg bases

For tables with different number of legs, we list all consistent combinations of legs and leg bases.

One legged

Two legged

Four legged

Special types

2. Textures

Wooden textures

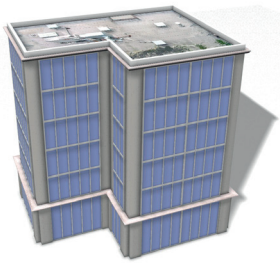
Steel textures

3. Table top styles

Rectangular top

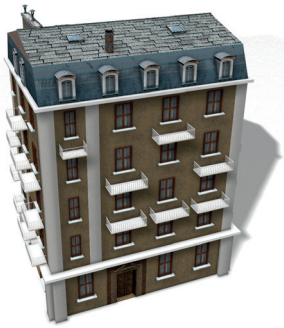
Round top

4. Building styles



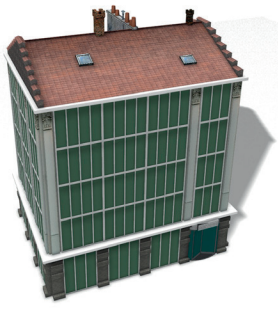
Office

R1



R2

R3



Mixed style

B Procedural Models Sampled According to the Designed PDFs

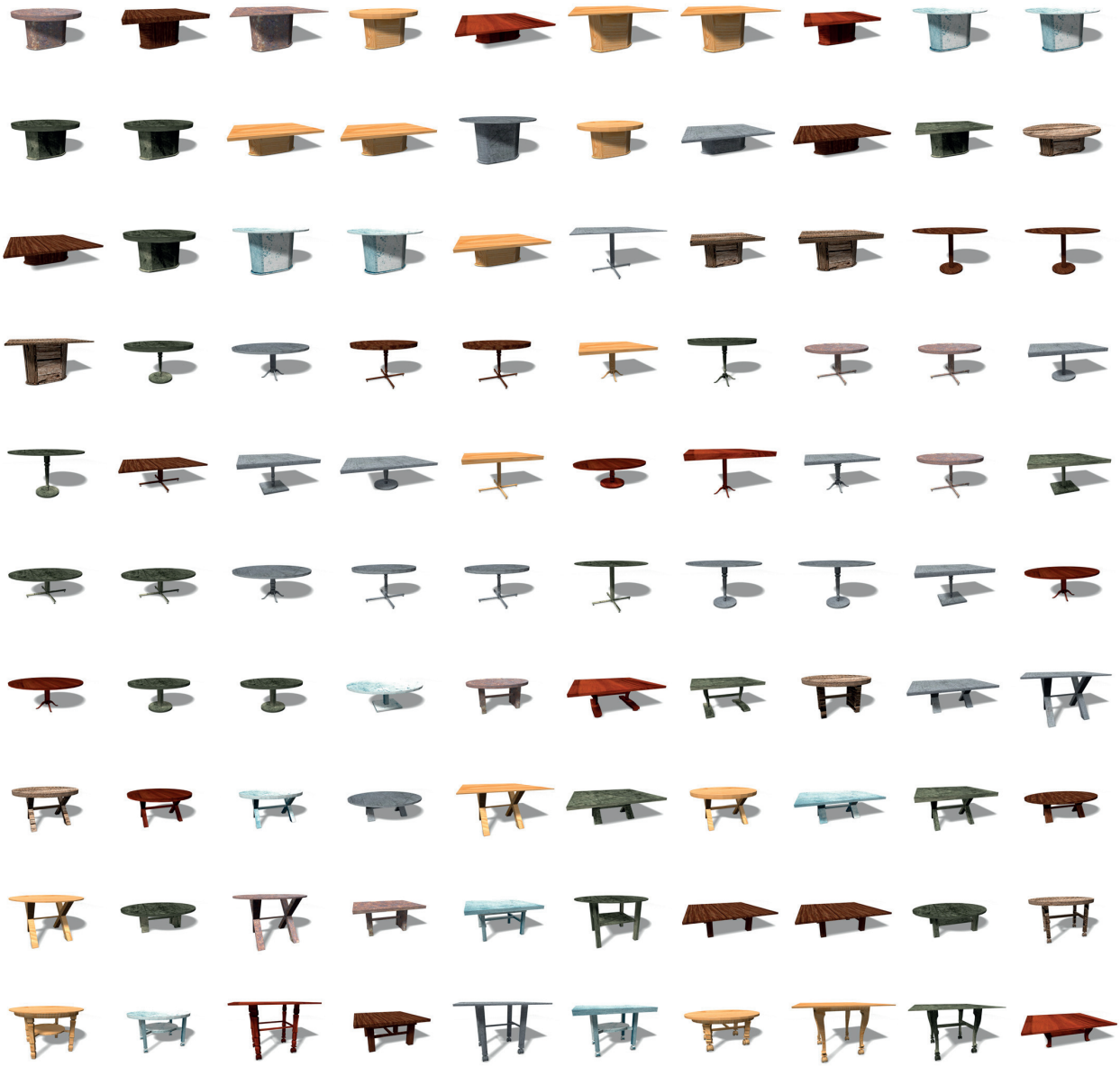
We provide 10×10 grids of models sampled from the designed density function for each design task in Section 4.7. We also include correlation scores to evaluate the correlation between the learned density function and the ground truth.

Design scenario F1

Target preferences:
Valid tables with one leg (70), two legs (20) and four legs (10). Non-valid tables (0).
A table is valid if the number of legs and the leg bases are consistent.

Example preference scores

Correlation w.r.t. ground truth



Design scenario F2

Target preferences:
Wooden tables with light color and round top: 60. Wooden tables with dark color and rectangular top: 40. Others: 0.

Example preference scores

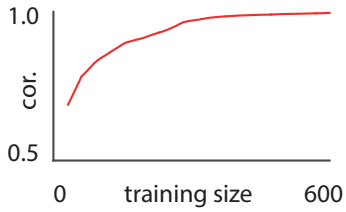
Correlation w.r.t. ground truth

Design scenario F3

Target preferences:
Valid tables with steel textures (70) and wooden textures (30). Non-valid tables (0).
A table is valid if the number of legs and the leg bases are consistent.

Example preference scores

Correlation w.r.t. ground truth

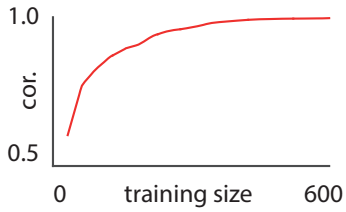


Design scenario F4

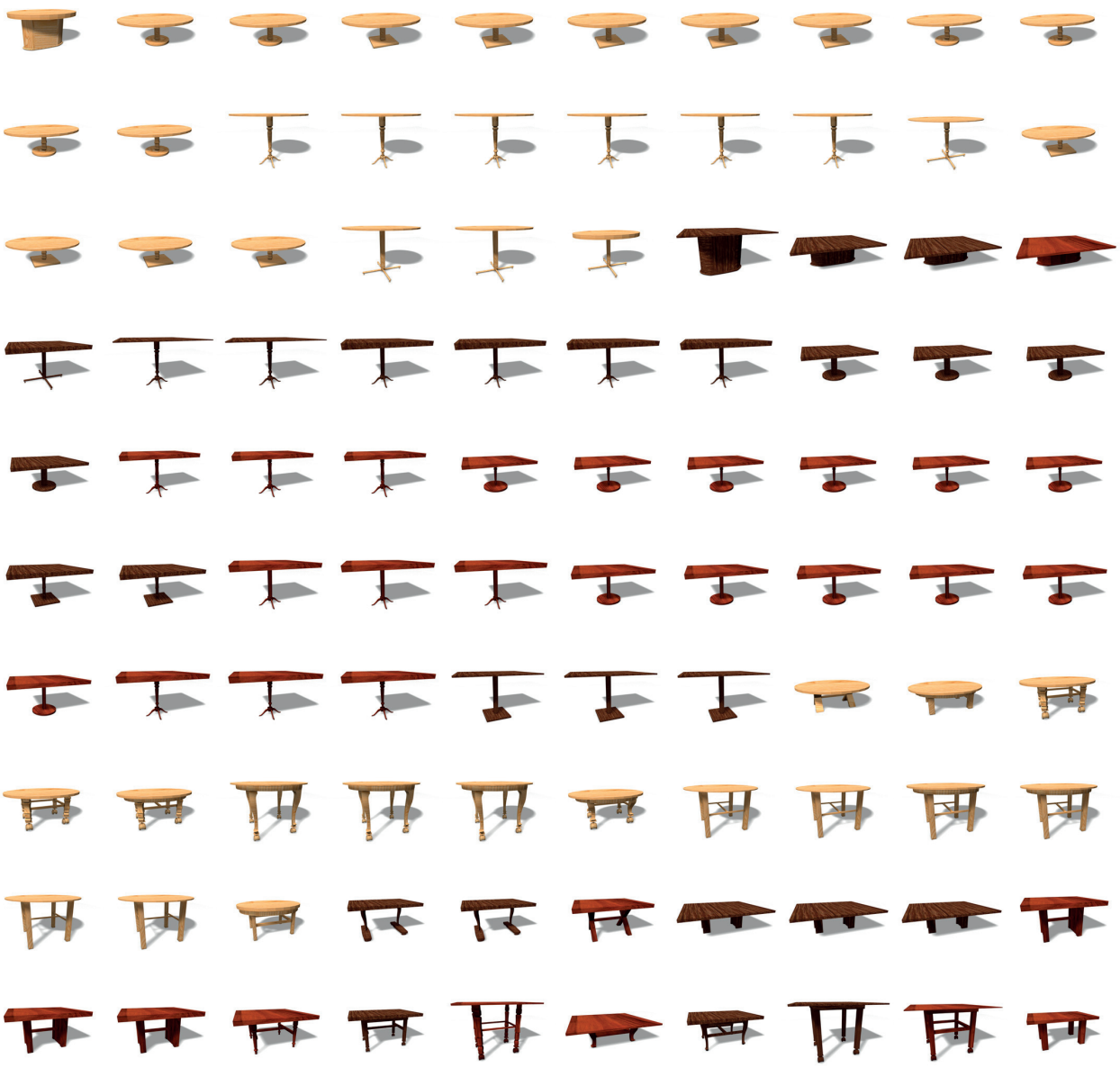
Target preferences:
Valid tables with round top (70) and rectangular top (30). Non-valid tables (0).
A table is valid if the number of legs and the leg bases are consistent.

Example preference scores

Correlation w.r.t. ground truth



Combination F1 x F2



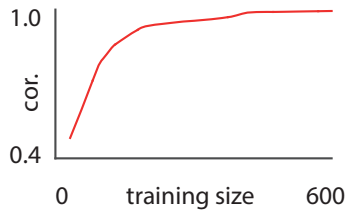
Combination F3 x F4

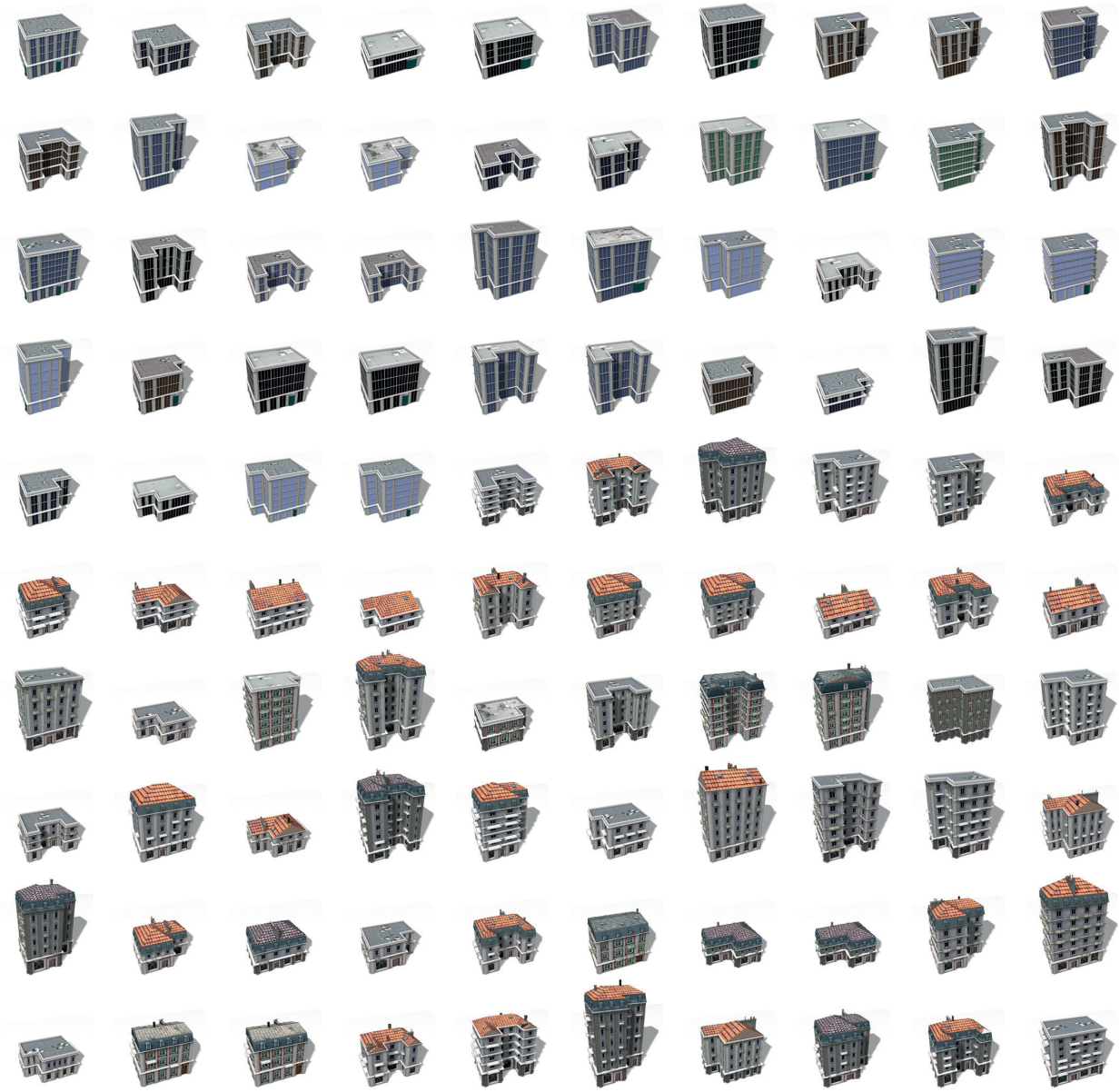
Design scenario B1

Target preferences:
Office: 40. Building style R1: 20. Building style R2: 20. Building style R3: 20. Mixed style: 0

Example preference scores

Correlation w.r.t. ground truth

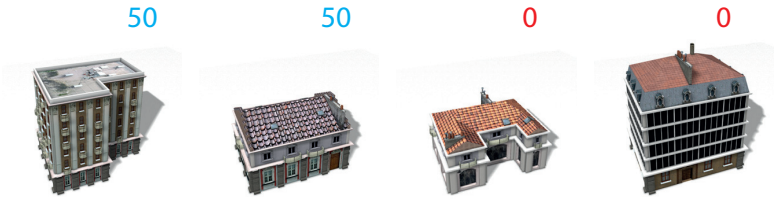




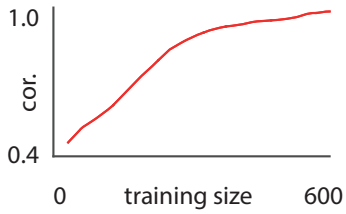
Design scenario B2

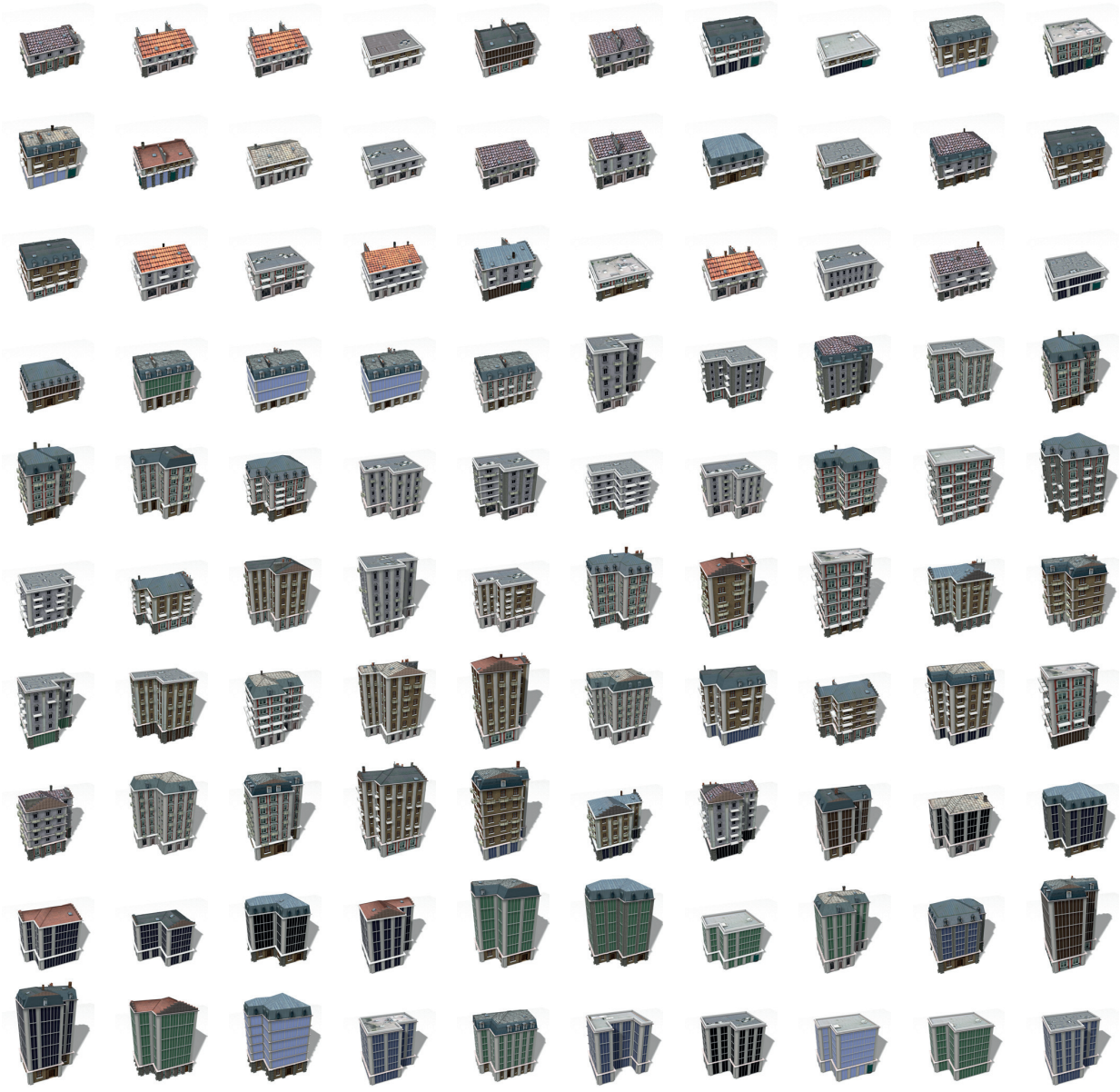
Target preferences:
Tall building (5 - 6 floors) and L-shape: 50.
Short building (2 - 3 floors) and rectangular shape: 50. Others: 0

Example preference scores



Correlation w.r.t. ground truth

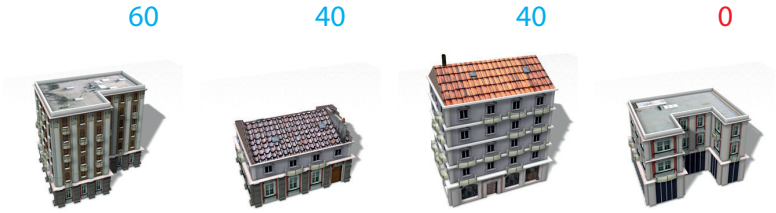




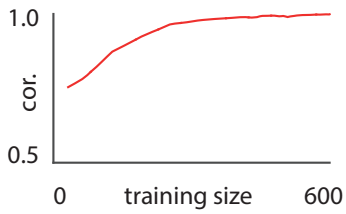
Design scenario B3

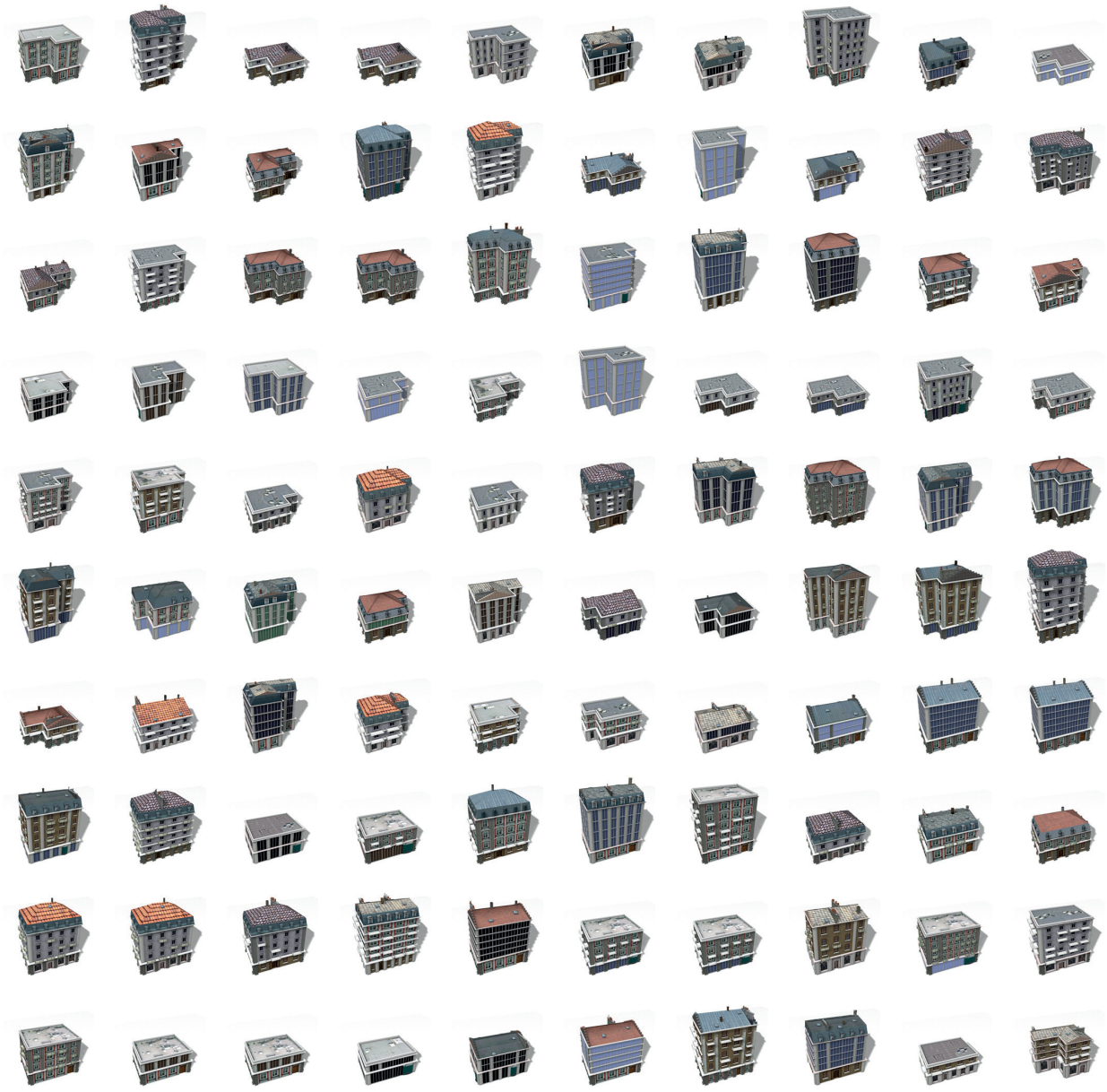
Target preferences:
L-shape: 60. Rectangular shape: 40. Others: 0

Example preference scores



Correlation w.r.t. ground truth



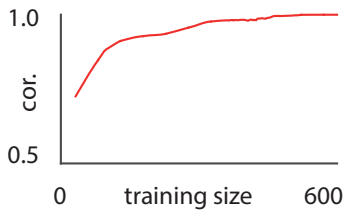


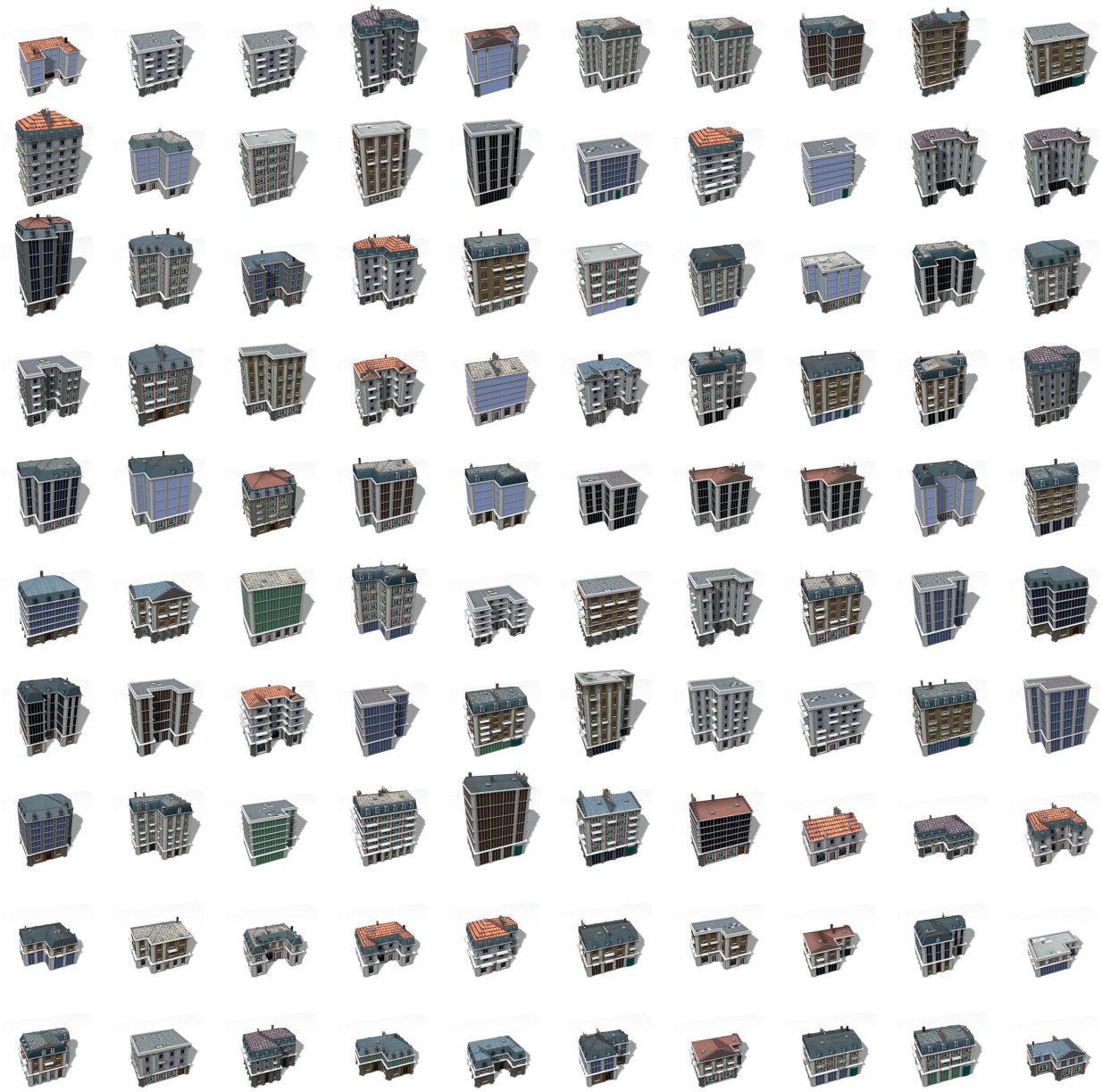
Design scenario B4

Target preferences:
Tall building (5 - 6 floors): 80. Short building (2 - 3 floors): 20

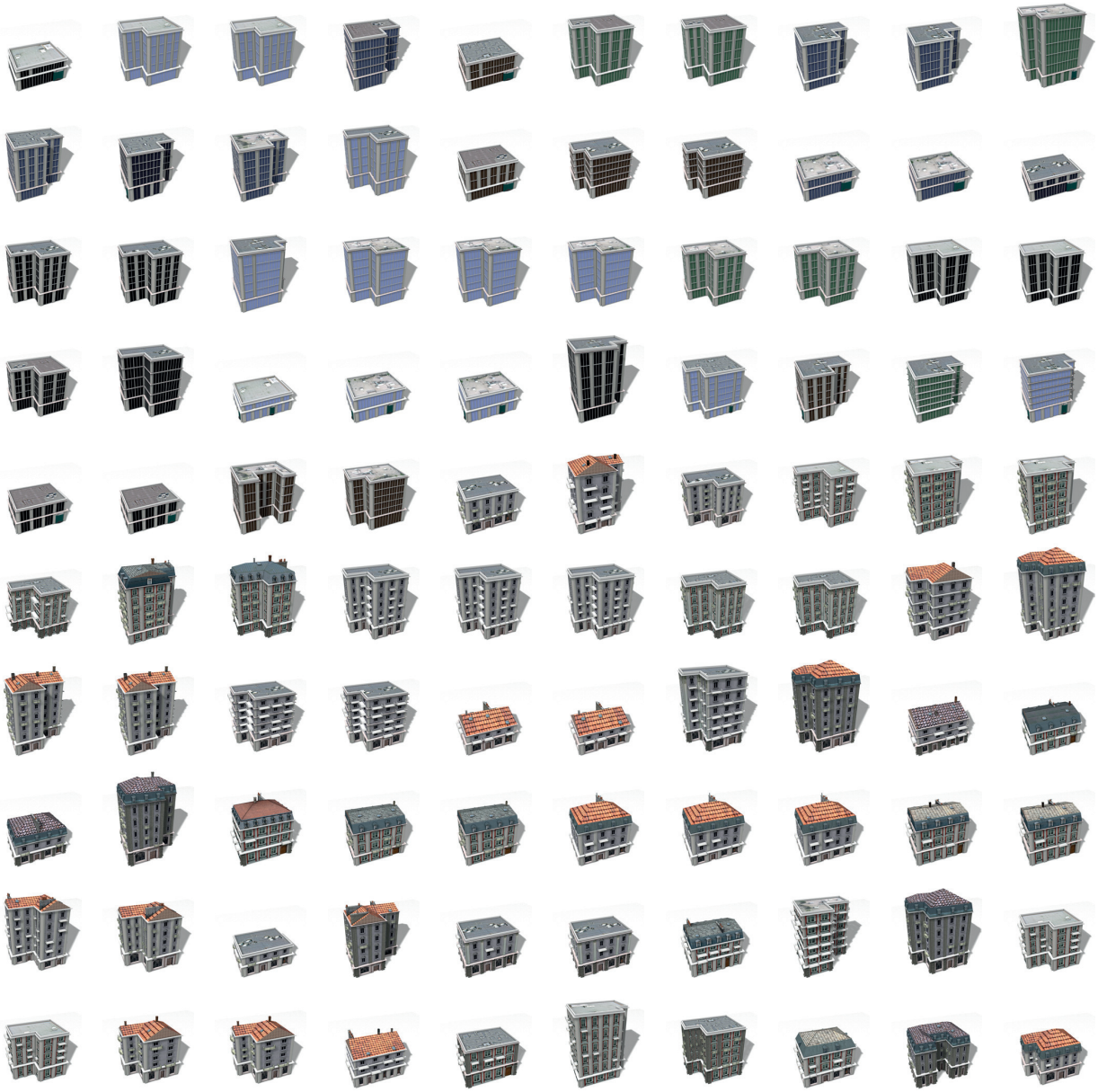
Example preference scores

Correlation w.r.t. ground truth

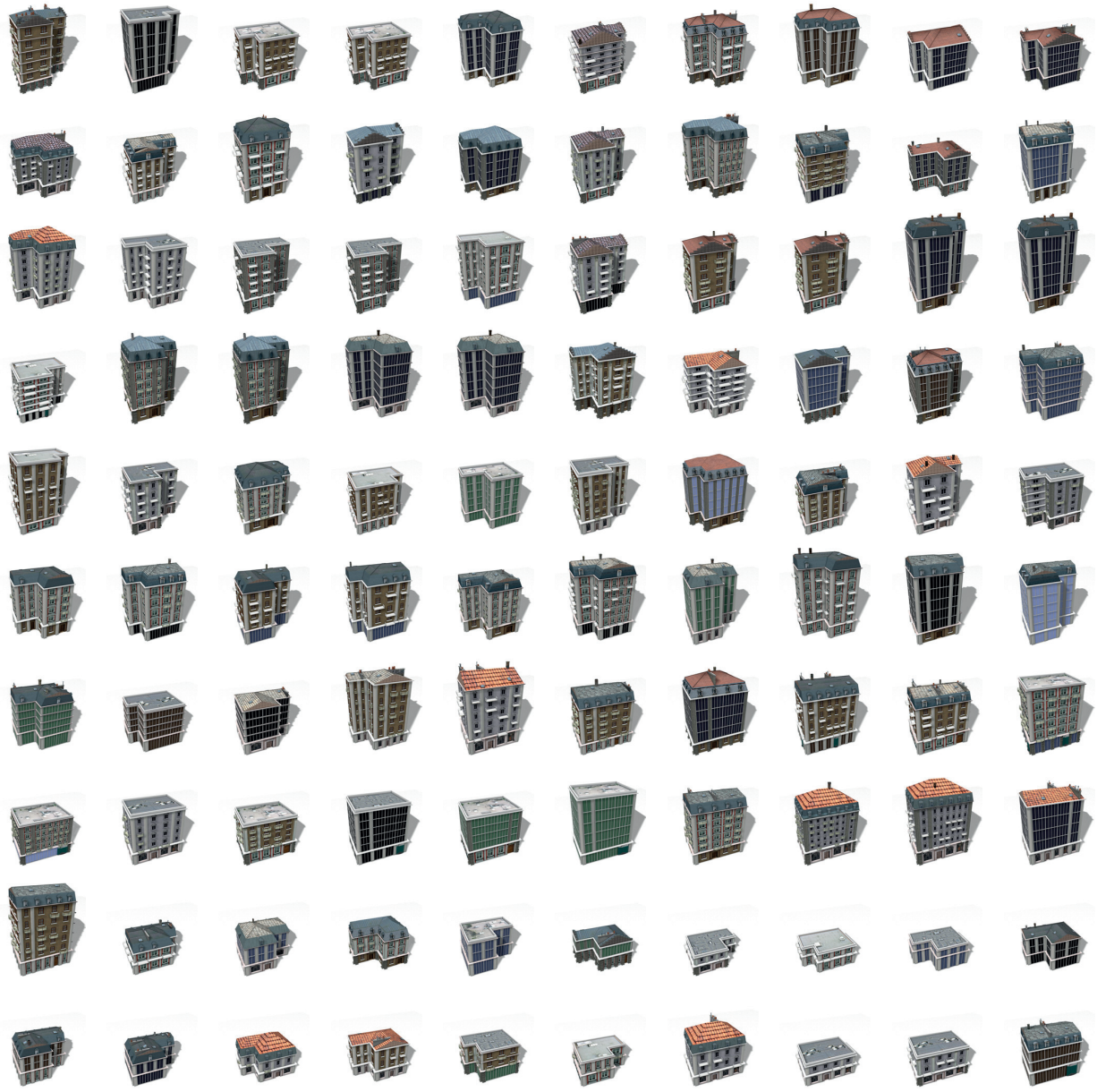




Combination B1 x B2



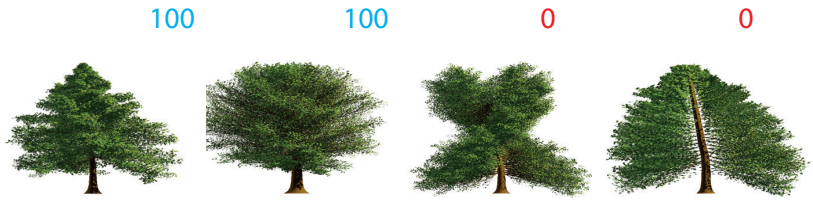
Combination B3 x B4



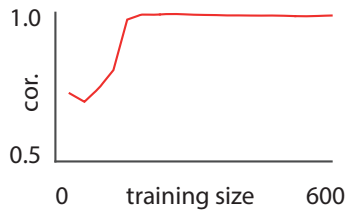
Design scenario T1

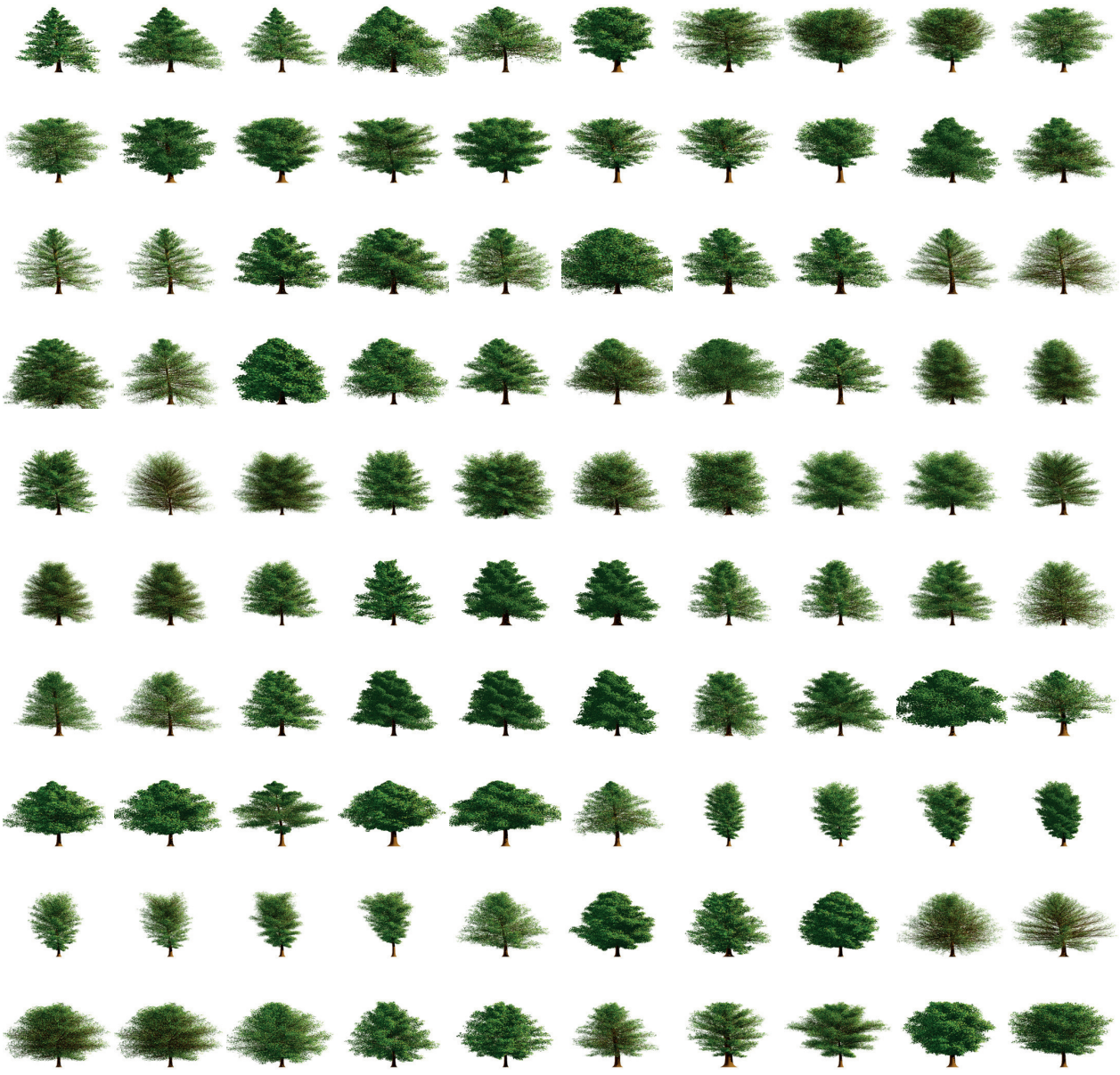
Target preferences:
Plausible tree: 100. Non-plausible tree: 0

Example preference scores



Correlation w.r.t. ground truth

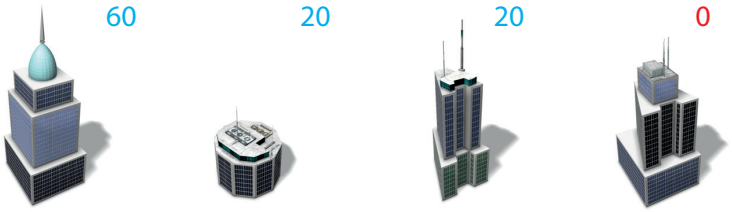




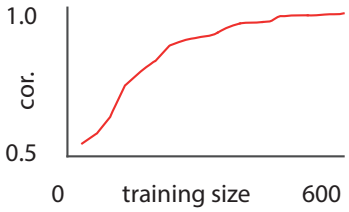
Design scenario S1

Target preferences:
Skyscrapers with only rectangular block: 60. Skyscrapers with only cylindrical blocks: 20.
Skyscrapers with only V-blocks: 20. Mixture of blocks: 0

Example preference scores



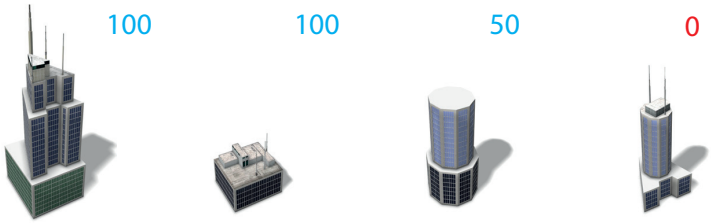
Correlation w.r.t. ground truth



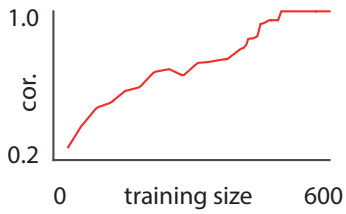
Design scenario S2

Target preferences:
Skyscrapers with rectangular base: 100, cylindrical base: 50, V-base: 0

Example preference scores



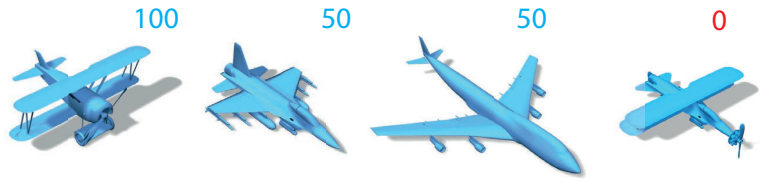
Correlation w.r.t. ground truth



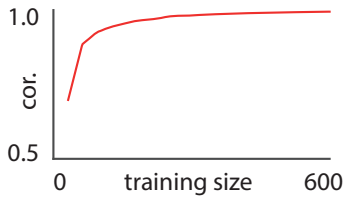
Design scenario A1

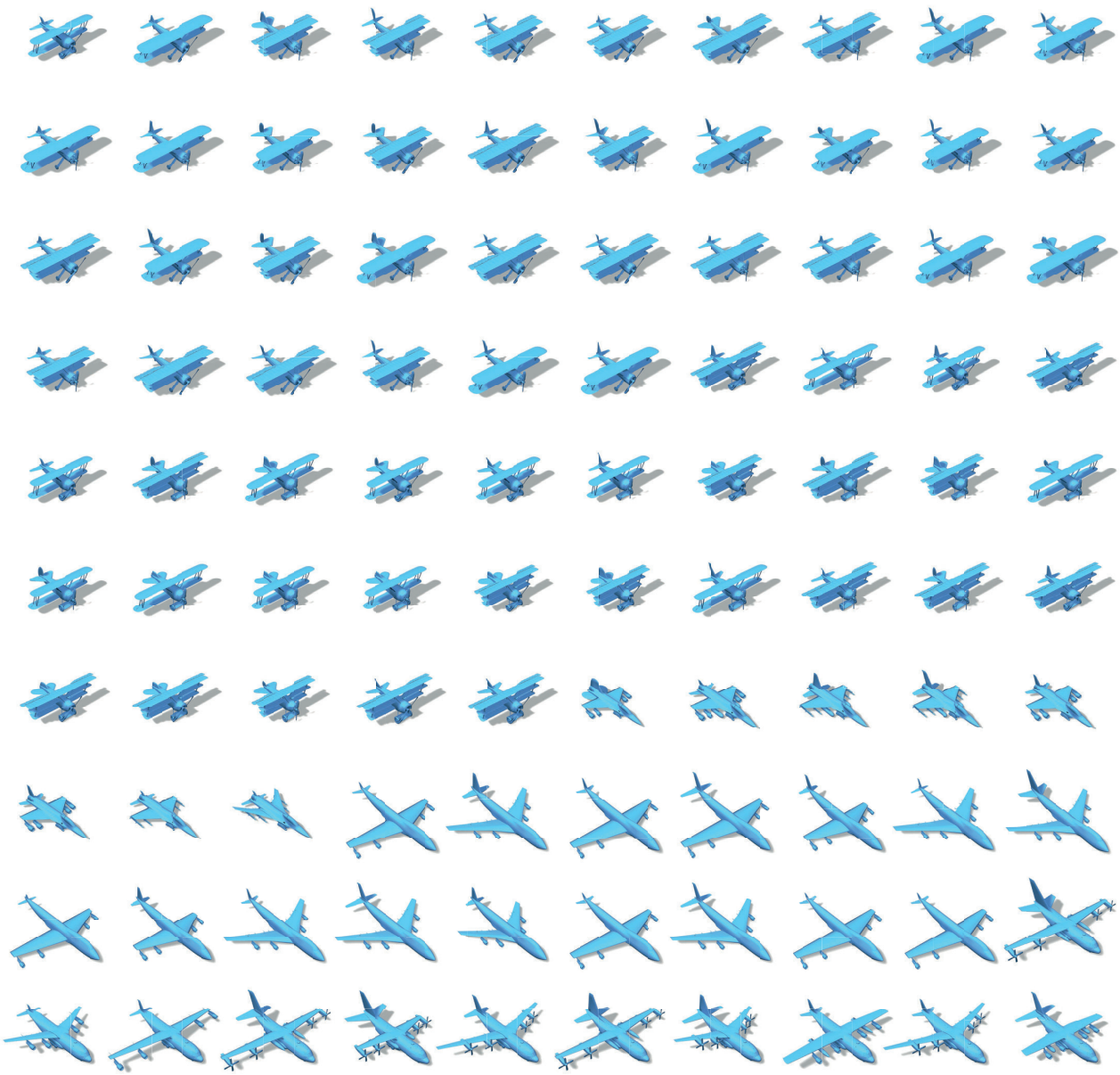
Target preferences:
Old-style airplanes: 100
Modern airplanes (commercial, transport airplanes, jet fighter): 50
Airplanes with mismatch components: 0

Example preference scores



Correlation w.r.t. ground truth





Bibliography

[1] Melinos Averkiou, Vladimir Kim, Youyi Zheng, and Niloy J. Mitra. ShapeSynth: Parameterizing Model Collections for Coupled Shape Exploration and Synthesis. *Comp. Graph. Forum (Eurographics)*, 33(2):125–134, 2014.

[2] Caroline Baillard, Cordelia Schmid, Andrew Zisserman, and Andrew Fitzgibbon. Automatic line matching and 3D reconstruction of buildings from multiple views. In *ISPRS Conf. on Automatic Extraction of GIS Objects from Digital Imagery*, pages 69–80, 1999.

[3] Fan Bao, Michael Schwarz, and Peter Wonka. Procedural facade variations from a single layout. *ACM Transactions on Graphics*, 32(1):8, 2013.

[4] Fan Bao, Dong-Ming Yan, Niloy J. Mitra, and Peter Wonka. Generating and Exploring Good Building Layouts. *ACM Trans. Graph. (Siggraph)*, 32(4):122, 2013.

[5] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[6] Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph. (Siggraph)*, 29(4):104, 2010.

[7] Martin Bokeloh, Michael Wand, Hans-Peter Seidel, and Vladlen Koltun. An algebraic model for parameterized shape editing. *ACM Trans. Graph. (Siggraph)*, page 78, 2012.

[8] Eric Brochu, Tyson Brochu, and Nando de Freitas. A Bayesian Interactive Optimization Approach to Procedural Animation Design. *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 103–112, 2010.

[9] Duygu Ceylan, Niloy J. Mitra, Hao Li, Thibaut Weise, and Mark Pauly. Fac-

tored facade acquisition using symmetric line arrangements. *Comp. Graph. Forum (Eurographics)*, 31:671–680, 2012.

[10] Duygu Ceylan, Niloy J. Mitra, Youyi Zheng, and Mark Pauly. Coupled structure-from-motion and 3D symmetry detection for urban facades. *ACM Transactions on Graphics*, 33(1):2:1–2:15, February 2014.

[11] Duygu Ceylan, Minh Dang, Niloy J Mitra, Boris Neubert, and Mark Pauly. Discovering structured variations via coupled template matching. *Comp. Graph. Forum*, 2015.

[12] Siddhartha Chaudhuri, Evangelos Kalogerakis, Stephen Giguere, and Thomas Funkhouser. Attribit: Content Creation with Semantic Attributes. *ACM Symp. User Interface Software and Technology*, pages 193–202, 2013.

[13] Noam Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124, 1956.

[14] Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *Advances in neural information processing systems*, pages 625–632, 2001.

[15] Minh Dang, Duygu Ceylan, Boris Neubert, and Mark Pauly. Safe: Structure-aware facade editing. *Comp. Graph. Forum (Eurographics)*, 33(2):83–93, 2014.

[16] Minh Dang, Stefan Lienhard, Duygu Ceylan, Boris Neubert, Peter Wonka, and Mark Pauly. Interactive design of probability density functions for shape grammars. *ACM Trans. Graph. (Siggraph Asia)*, 34(6):206, 2015.

[17] Andrew Delong, Anton Osokin, Hossam N Isack, and Yuri Boykov. Fast approximate energy minimization with label costs. *Int. Journal on Computer Vision*, 96(1):1–27, 2012.

[18] Bailin Deng, Sofien Bouaziz, Mario Deuss, Juyong Zhang, Yuliy Schwartzburg, and Mark Pauly. Exploring Local Modifications for Constrained Meshes. *Comp. Graph. Forum (Eurographics)*, 32(2pt1):11–20, 2013.

[19] Anthony R Dick, Philip HS Torr, Simon J Ruffle, and Roberto Cipolla. Combining single view recognition and multiple view stereo for architectural scenes. In *IEEE ICCV*, volume 1, pages 268–274. IEEE, 2001.

[20] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei A. Efros. What makes paris look like paris? *ACM Trans. Graph. (Siggraph)*, 31(4):101, 2012.

[21] Xiaowen Dong, Pascal Frossard, Pierre Vandergheynst, and Nikolai Nefedov. Clustering on multi-layer graphs via subspace analysis on grassmann manifolds. *IEEE Trans. on Signal Processing*, 62(4):905–918, 2014.

[22] Frances Downing and Ulrich Flemming. *The bungalows of buffalo*. Department of Architecture, Carnegie-Mellon University, 1981.

[23] J Duarte. *Malagueira Grammar–towards a tool for customizing Alvaro Siza's mass houses at Malagueira*. PhD thesis, MIT School of Architecture and Planning, 2002.

[24] Alon Faktor and Michal Irani. Co-segmentation by composition. In *IEEE ICCV*, pages 1297–1304. IEEE, 2013.

[25] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Describing Objects By Their Attributes. *IEEE CVPR*, pages 1778–1785, 2009.

[26] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE PAMI*, 32:1362–1376, 2009.

[27] Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. iWIRES: An analyze-and-edit approach to shape manipulation. *ACM Trans. Graph. (Siggraph)*, 28:33, 2009.

[28] James Gips. *Shape grammars and their uses*. PhD thesis, Stanford University, 1974.

[29] Nianjuan Jiang, Ping Tan, and Loong-Fah Cheong. Multi-view repetitive structure detection. In *IEEE ICCV*, pages 535–542, 2011. ISBN 978-1-4577-1101-5.

[30] Mark Johnson. Pcfg models of linguistic tree representations. *Comput. Linguist.*, 24 (4):613–632, 1998.

[31] Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. (Siggraph)*, 31(4):55, 2012.

[32] William B. Kerr and Fabio Pellacini. Toward Evaluating Material Design Interface Paradigms for Novice Users. *ACM Trans. Graph. (Siggraph)*, 29(4):35, 2010.

[33] Yanir Kleiman, Noa Fish, Joel Lanir, and Daniel Cohen-Or. Dynamic Maps for Exploring and Browsing Shapes. *Comp. Graphics Forum (SGP)*, 32(5):187–196, 2013.

[34] Panagiotis Koutsourakis, Loic Simon, Olivier Teboul, Georgios Tziritas, and Nikos Paragios. Single view reconstruction using shape grammars for urban environments. In *IEEE ICCV*, pages 1795–1802, 2009.

[35] Lucas Kovar and Michael Gleicher. Simplicial Families of Drawings. *ACM Symp. User Interface Software and Technology*, pages 163–172, 2001.

[36] Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. Crowd-powered Parameter Analysis for Visual Design Exploration. *ACM Symp. User Interface Software and Technology*, pages 65–74, 2014.

[37] Lars Krecklau, Darko Pavic, and Leif Kobbelt. Generalized use of non-terminal symbols for procedural modeling. *Comp. Graph. Forum (Eurographics)*, 29(8): 2291–2303, 2010.

[38] Christian Kurz, Xiaokun Wu, Michael Wand, Thorsten Thormählen, P Kohli, and H-P Seidel. Symmetry-aware template deformation and fitting. *Comp. Graph. Forum*, 33(6):205–219, 2014.

[39] Florent Lafarge, Renaud Keriven, Mathieu Brédif, and Vu Hiep. A hybrid multi-view stereo algorithm for modeling urban scenes. *IEEE PAMI*, 35(1):5–17, January 2013.

[40] Erik G Learned-Miller. Data driven image models through continuous joint alignment. *IEEE PAMI*, 28(2):236–250, 2006.

[41] Brian Lee, Savil Srivastava, Ranjitha Kumar, Ronen I. Brafman, and Scott R. Klemmer. Designing with Interactive Example Galleries. *Proc. SIGCHI Conf. on Human Factors in Comp. Sys.*, pages 2257–2266, 2010.

[42] Sylvain Lefebvre, Samuel Hornus, and Anass Lasram. By-example synthesis of architectural textures. *ACM Trans. Graph. (Siggraph)*, 29(4):84, 2010.

[43] George Leifman and Avishay Tal. Pattern-driven colorization of 3D surfaces. In *IEEE CVPR*, pages 241–248, June 2013.

[44] Yangyan Li, Xiaokun Wu, Yiorgos Chrysanthou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra. Globfit: Consistently fitting primitives by discovering global relations. *ACM Trans. Graph. (Siggraph)*, 30(4):52:1–52:12, 2011.

[45] Yangyan Li, Qian Zheng, Andrei Sharf, Daniel Cohen-Or, Baoquan Chen, and Niloy J. Mitra. 2D-3D fusion for layer decomposition of urban facades. In *IEEE ICCV*, 2011.

[46] Stefan Lienhard, Matthias Specht, Boris Neubert, Mark Pauly, and Pascal Müller. Thumbnail Galleries for Procedural Models. *Comp. Graph. Forum (Eurographics)*, 33(2):361–370, 2014.

[47] Jinjie Lin, Daniel Cohen-Or, Hao (Richard) Zhang, Cheng Liang, Andrei Sharf, Oliver Deussen, and Baoquan Chen. Structure-preserving retargeting of irregular 3D architecture. *ACM Trans. Graph. (Siggraph Asia)*, 30(6):183, 2011.

[48] Yaron Lipman, Xiaobai Chen, Ingrid Daubechies, and Thomas Funkhouser. Symmetry factored embedding and distance. *ACM Trans. Graph. (Siggraph)*, 29(4): 103:1–103:12, July 2010.

[49] Shenglan Liu, Ralph R. Martin, Frank C. Langbein, and Paul L. Rosin. Segmenting periodic reliefs on triangle meshes. In *Proc. of the IMA Int. Conf. on Math. of Surfaces*, pages 290–306, 2007.

[50] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[51] Joe Marks, Brad Andalman, Paul A Beardsley, William Freeman, Sarah Gibson, Jessica Hodgins, Thomas Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, et al. Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation. *Proc. of SIGGRAPH*, pages 389–400, 1997.

[52] Andelo Martinovic and Luc Van Gool. Bayesian grammar learning for inverse procedural modeling. *IEEE CVPR*, pages 201–208, 2013.

[53] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive Furniture Layout Using Interior Design Guidelines. *ACM Trans. Graph. (Siggraph)*, 30(4):87, 2011.

[54] Niloy J Mitra, Leonidas J Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph. (Siggraph)*, 25(3):560–568, 2006.

[55] Niloy J. Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. Symmetry in 3d geometry: Extraction and applications. *Comp. Graph. Forum*, 32(6):1–23, 2013.

[56] Niloy J Mitra, Michael Wand, Hao Richard Zhang, Daniel Cohen-Or, and Martin Bokeloh. Structure-aware shape processing. *Comp. Graph. Forum*, pages 175–197, 2013.

[57] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural Modeling of Buildings. *ACM Trans. Graph. (Siggraph)*, 25(3):614–623, 2006.

[58] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. *ACM Trans. Graph. (Siggraph)*, 26(3):85, 2007.

[59] Przemyslaw Musialski, Michael Wimmer, and Peter Wonka. Interactive coherence-based façade modeling. *Comp. Graph. Forum*, 31(2):661–670, 2012.

[60] Przemyslaw Musialski, Peter Wonka, Daniel G Aliaga, Michael Wimmer, L Gool, and Werner Purgathofer. A survey of urban reconstruction. *Comp. Graph. Forum*, 32(6):146–177, 2013.

[61] Radomír Měch and Przemyslaw Prusinkiewicz. Visual Models of Plants Interacting with Their Environment. *Proc. of SIGGRAPH*, pages 397–410, 1996.

[62] Liangliang Nan, Andrei Sharf, Ke Xie, Tien-Tsin Wong, Oliver Deussen, Daniel Cohen-Or, and Baoquan Chen. Conjoining gestalt rules for abstraction of architectural drawings. *ACM Trans. Graph. (Siggraph Asia)*, 30(6):185, 2011.

[63] Liangliang Nan, Caigui Jiang, Bernard Ghanem, and Peter Wonka. Template assembly for detailed urban reconstruction. *Comp. Graph. Forum (Eurographics)*, 34(2):217–228, 2015.

[64] Peter O'Donovan, Jānis Lībeks, Aseem Agarwala, and Aaron Hertzmann. Exploratory Font Selection Using Crowdsourced Attributes. *ACM Trans. Graph. (Siggraph)*, 33(4):92, 2014.

[65] Yoav I. H. Parish and Pascal Müller. Procedural Modeling of Cities. *Proc. of SIGGRAPH*, pages 301–308, 2001.

[66] Mark Pauly, Niloy J Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J Guibas. Discovering structural regularity in 3D geometry. *ACM Trans. Graph. (Siggraph)*, 27(3):43, 2008.

[67] John C Platt, Christopher JC Burges, Steven Swenson, Christopher Weare, and Alice Zheng. Learning a gaussian process prior for automatically generating music playlists. In *NIPS*, pages 1425–1432, 2001.

[68] Przemyslaw Prusinkiewicz. Graphical Applications of L-systems. *Proc. on Graphics Interface/Vision Interface*, 86(1986):247–253, 1986.

[69] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer, 1990.

[70] Przemyslaw Prusinkiewicz, Mark James, and Radomír Měch. Synthetic Topiary. *Proc. of SIGGRAPH*, pages 351–358, 1994.

[71] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. MIT Press, 2005.

[72] Hayko Riemenschneider, Ulrich Krispel, Wolfgang Thaller, Michael Donoser, Sven Havemann, Dieter Fellner, and Horst Bischof. Irregular lattices for complex shape grammar facade parsing. In *IEEE CVPR*, pages 1640–1647, 2012.

[73] Konrad Schindler. A model-based method for building reconstruction. In *Proc. of the Int. Conf. on Comp. Vis. Work. on Higher-Level Know. in 3D Model. & Motion*, pages 74–82, 2003.

[74] Ruwen Schnabel, Patrick Degener, and Reinhard Klein. Completion and reconstruction with primitive shapes. *Comp. Graph. Forum (Eurographics)*, 28(2):503–512, 2009.

[75] Michael Schwarz and Pascal Müller. Advanced probcedural modeling of architecture. *ACM Trans. Graph. (Siggraph)*, 34(4):107, 2015.

[76] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Image Appearance Exploration by Model-Based Navigation. *Comp. Graph. Forum (Eurographics)*, 28(2):629–638, 2009.

[77] Eli Shechtman and Michal Irani. Matching local self-similarities across images and videos. In *IEEE CVPR*, pages 1–8, June 2007.

[78] Chao-Hui Shen, Shi-Sheng Huang, Hongbo Fu, and Shi-Min Hu. Adaptive partitioning of urban facades. *ACM Trans. Graph. (Siggraph Asia)*, 30(6):184, 2011.

[79] Loic Simon, Olivier Teboul, Panagiotis Koutsourakis, and Nikos Paragios. Random Exploration of the Procedural Space for Single-View 3D Modeling of Buildings. *IJCV*, 93(2):253–271, 2011.

[80] Ruben M. Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. A Survey on Procedural Modelling for Virtual Worlds. *Comp. Graph. Forum (Eurographics)*, 33 (6):31–50, 2014.

[81] O. Šťava, B. Beneš, R. Měch, D. G. Aliaga, and P. Krištof. Inverse procedural modeling by automatic generation of l-systems. *Comp. Graph. Forum (Eurographics)*, 29(2):665–674, 2010.

[82] George Stiny. *Pictorial and Formal Aspects of Shape and Shape Grammars*. Birkhauser Verlag, Basel, Switzerland, 1975.

[83] George Stiny. Spatial Relations and Grammars. *Environment and Planning B*, 5(1): 5–18, 1982.

[84] George Stiny, William J Mitchell, et al. The palladian grammar. *Environment and planning B*, 5(1):5–18, 1978.

[85] Jerry Talton, Daniel Gibson, Lingfeng Yang, Pat Hanrahan, and Vladlen Koltun. Exploratory Modeling with Collaborative Design Spaces. *ACM Trans. Graph. (Siggraph Asia)*, 28(5):167, 2009.

[86] Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomír Měch. Learning design patterns with bayesian grammar induction. *ACM Symp. User Interface Software and Technology*, pages 63–74, 2013.

[87] Olivier Teboul, Loic Simon, Panagiotis Koutsourakis, and Nikos Paragios. Segmentation of building facades using procedural shape priors. In *IEEE CVPR*, pages 3105–3112, 2010.

[88] Olivier Teboul, Iasonas Kokkinos, Loic Simon, Panagiotis Koutsourakis, and Nikos Paragios. Shape grammar parsing via reinforcement learning. In *IEEE CVPR*, pages 2273–2280, 2011.

[89] Nobuyuki Umetani, Takeo Igarashi, and Niloy J. Mitra. Guided Exploration of Physically Valid Shapes for Furniture Design. *ACM Trans. Graph. (Siggraph)*, 58 (9):86, 2012.

[90] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? In *ICML*, pages 1073–1080, 2009.

[91] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant architecture. *ACM Trans. Graph. (Siggraph)*, 22(3):669–677, 2003.

[92] Changchang Wu. Towards linear-time incremental structure from motion. In *3DV-Conference*, pages 127–134, 2013.

[93] Changchang Wu, Jan-Michael Frahm, and Marc Pollefeys. Repetition-based dense single-view reconstruction. In *IEEE CVPR*, 2011.

[94] Fuzhang Wu, Dong-Ming Yan, Weiming Dong, Xiaopeng Zhang, and Peter Wonka. Inverse Procedural Modeling of Facade Layouts. *ACM Trans. Graph. (Siggraph)*, 33 (4):121, 2014.

[95] Kai Xu, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Fit and Diverse: Set Evolution for Inspiring 3D Shape Galleries. *ACM Trans. Graph. (Siggraph)*, 31(4): 57, 2012.

[96] Yong-Liang Yang, Yi-Jun Yang, Helmut Pottmann, and Niloy J. Mitra. Shape Space Exploration of Constrained Meshes. *ACM Trans. Graph. (Siggraph Asia)*, 30(6): 124, 2011.

[97] Hao Zhang, Kai Xu, Wei Jiang, Jinjie Lin, Daniel Cohen-Or, and Baoquan Chen. Layered analysis of irregular facades via symmetry maximization. *ACM Trans. Graph. (Siggraph)*, 32(4):121, 2013.

[98] Qian Zheng, Andrei Sharf, Guowei Wan, Yangyan Li, Niloy J. Mitra, Daniel Cohen-Or, and Baoquan Chen. Non-local scan consolidation for 3D urban scenes. *ACM Trans. Graph. (Siggraph)*, 29(4):94, 2010.

Minh DANG

Chbr. 4, Rte. Cantonale 35, St-Sulpice, Vaud, Switzerland CH-1025
+41 (0) 77 435 46 49 | minh.dang@epfl.ch | http://nmdang.com

Profile

- Computer graphics researcher interested in 3D reconstruction and procedural modeling.

Education

- Sep 2010 - Jan 2016 (expected) — **Ph.D. in Computer Science**
  Computer Graphics and Geometry Laboratory (LGG)
  Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
- Nov 2014 - Jan 2015 — **Visiting Ph.D. Student**
  Visual Computing Center (VCC)
  King Abdullah University of Science and Technology (KAUST), Saudi Arabia
- Aug 2006 - May 2010 — **B.Eng. in Electrical and Electronic Engineering**
  Nanyang Technological University (NTU), Singapore

Work experience

- Jun 2015 - Sep 2015 — **Software Engineering Intern, Google Inc.**
  Worked in a research project of Google Maps.
- Jan 2011 - to date — **Research and Teaching Assistant, Computer Graphics and Geometry Laboratory (LGG), EPFL, Switzerland**
  Project: Computational Symmetry for Geometry Data Analysis and Design
  Course taught: CS-440 Advanced computer graphics, CS-446 Digital 3D geometry processing, CS-341 Introduction to computer graphics
  Other activities: Advising Bachelor's and Master's students on theses and semester projects

Publications

- **Discovering structured variations via coupled template matching**
  Duygu Ceylan, Minh Dang, Niloy Mitra, Boris Neubert, Mark Pauly
  Computer Graphics Forum 2015
  Detecting similarities in noisy and incomplete 3D point clouds is extremely difficult. This system enables that via the use of templates and progressively improves the detection quality by iterating between similarity detection and template matching.
  Potential use: use the detected similarities to improve the quality of 3D reconstruction.
- **Interactive design of probability density functions for shape grammars** [1]
  Minh Dang, Stefan Lienhard, Duygu Ceylan, Boris Neubert, Peter Wonka, Mark Pauly
  ACM Transactions on Graphics (SIGGRAPH Asia) 2015
  While automatic mass generation of models (e.g. the generation of thousands of buildings) is enabled by the procedural modeling paradigm, it is difficult to control. This framework makes procedural modeling flexible and adapted to user preferences. The user preferences are learned by active learning.
  Potential use: generate buildings for 3D cities.

[1] http://go.epfl.ch/proman

- **SAFE: Structure-aware facade editing** [2]
  Minh Dang, Duygu Ceylan, Boris Neubert, Mark Pauly
  Computer Graphics Forum (EUROGRAPHICS) 2014
  Edit building facades while preserving their structures. Handle facades with irregular structures.
  Potential use: generate high quality textures for 3D cities.

- **A novel approach to remove redundant Gabor wavelets for family classification**
  Mohammad Ghahraman, Minh Dang, Wei-Yun Yau, Eam-Khwang Teoh
  International Conference on Control, Automation, Robotics and Vision (ICARCV) 2010
  Select relevant features from face images to spot family resemblance.
  Potential use: reunite lost family members from images in social networks.

Professional skills

- Technical skills: Computer Graphics, Procedural Modeling, 3D Reconstruction, Computational Geometry, Computer Vision, Machine Learning, Image Processing
- Programming skills: Operating systems: Mac OS X, Linux, Windows
  Languages: C/C++, Matlab, Python
  Frameworks: OpenGL, OpenCV, OpenMesh, Qt, Ipopt, Gurobi
- Professional tools: Adobe Photoshop | Illustrator | After Effects, Autodesk Maya, Esri CityEngine
- Language skills: Vietnamese (native), English (fluent)

Awards

- Sep 2010 - Sep 2011 **EDIC Fellowship (merit-based)**
  To finance the first year of Ph.D. program in EPFL
- Aug 2008 - May 2009 **NTU President Research Scholar**
  For participants of the Undergraduate Research Experience on Campus (URECA)
- Aug 2007 - May 2009 **Dean's List, Nanyang Technological University**
  Top 5% of the student cohort
- Aug 2006 - May 2010 **Singapore Scholarship (merit-based)**
  To finance the undergraduate study in Singapore

Other activities

- Reviewer, SIGGRAPH Asia 2014, Pacific Graphics 2015, Computer & Graphics
- Executive committee member, NTU Alumni Association in Europe
- Hobbies: Kendo, Taekwondo, Hiking, Sailing

[2] `http:/go.epfl.ch/safe-eg`