Structure Completion for Facade Layouts

Lubin Fan[1,2] Przemyslaw Musialski[3] Ligang Liu[4] Peter Wonka[1,5]

[1]KAUST [2]Zhejiang University [3]TU Vienna [4]USTC [5]Arizona State University

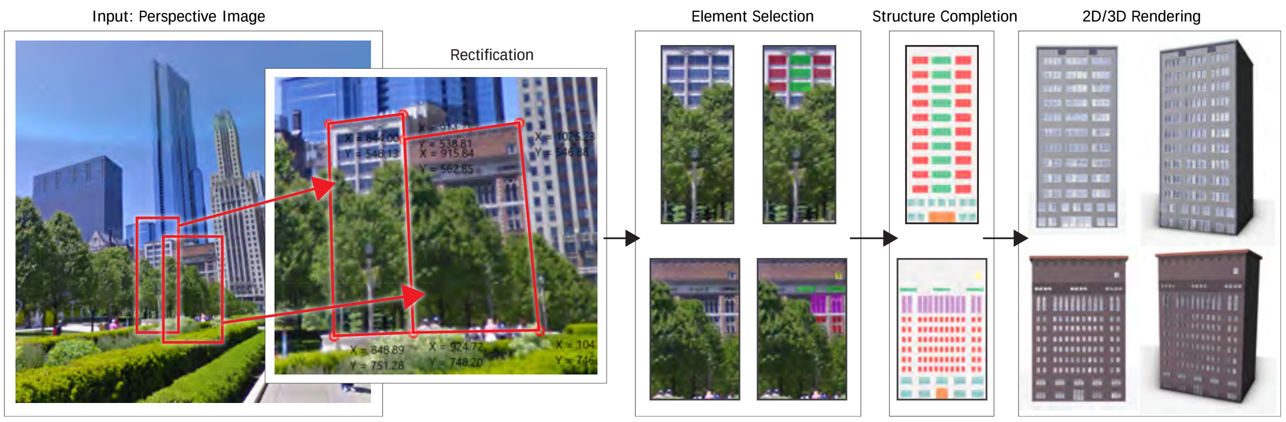

Figure 1: *Starting from a single image (Input) a user can specify a building mass model and mark shapes (windows, doors, ornaments) on the observed part of a building facade in a rectified image (Element Selection). Our structure completion framework can complete the missing part of the layout (Structure Completion). A 2D and a 3D rendering of the completed models is shown on the right.*

Abstract

We present a method to complete missing structures in facade layouts. Starting from an abstraction of the partially observed layout as a set of shapes, we can propose one or multiple possible completed layouts. Structure completion with large missing parts is an ill-posed problem. Therefore, we combine two sources of information to derive our solution: the observed shapes and a database of complete layouts. The problem is also very difficult, because shape positions and attributes have to be estimated jointly. Our proposed solution is to break the problem into two components: a statistical model to evaluate layouts and a planning algorithm to generate candidate layouts. This ensures that the completed result is consistent with the observation and the layouts in the database.

CR Categories: I.3.8 [Computing Methodologies]: Computer Graphics—Applications

Keywords: structure completion, facade modeling, urban reconstruction, inpainting

Links: DL PDF WEB VIDEO DATA

1 Introduction

We are interested in the problem of completing a layout of shapes from a partial observation. The main motivating application for this work is urban reconstruction, where building faces are often partially occluded by other buildings or vegetation. Our framework can help to reconstruct missing parts of the building (see Figure 1). Because it is often difficult to capture a complete facade in a single image there exist an abundance of incomplete facade images, e.g. from Google Street-View or Flickr, and there is great demand to complete them in an urban modeling application.

The problem of structure completion can be interpreted as an extension to image completion. First approaches emphasized smoothness as an important quality criterion to complete smaller scratches in an image [Bertalmío et al. 2000]. Later approaches also included mechanisms to enforce edge continuity, e.g. [Sun et al. 2005] and most recent methods can continue repetitions, if they detect a few existing elements, e.g. [He and Sun 2012; Dai et al. 2013]. We call these approaches local structure completion, because they require that each element in a layout is only concerned with its immediate neighbors. A great way to formulate local structure completion is to use an MRF formulation. Such a formulation can work well to estimate the label of one or a few missing elements, but a suitable heuristic for element positions has to be available. By contrast, our approach tries to model and complete structures where a global analysis of the data is necessary. We would like to complete structures where half or more of the layout is not visible. See Figure 2 for an example why structure completion is difficult.

One challenge in structure completion is that it is ill posed as there exist many plausible solutions, especially if a large amount of information is missing. It is not sufficient to repeat the observed elements in the layout using the observed spacing from the layout. Our approach to this challenge is to model probability distributions of layouts and learning these distributions from a database using graphical models. A second challenge in structure completion is that the number of missing shapes as well as their positions and attributes have to be estimated. Estimating the positions is the most

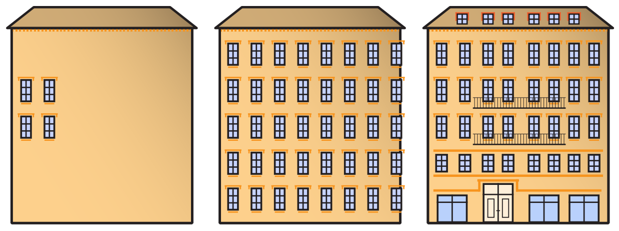

Figure 2: *Challenges of structure completion. Left: an input domain with only few observed elements. Middle: it is not sufficient to complete grids and to repeat elements using observed spacings. The result is too simple and leads to many inconsistencies, e.g. bad spacing with respect to a boundary as shown here. Right: A good structure completion result needs to account for many subtle irregularities in element position and also use additional knowledge, e.g. to place a door and shop windows in the first floor.*

difficult aspect of the problem. In our solution we propose a planning algorithm that can generate completion candidates by jointly computing the position and attributes of the elements.

The main contributions of our work are:

- We pose the problem of global structure completion for facade layouts.
- We propose a first solution to this problem and show that our solution can complete facades with only a small number of observed elements.
- We demonstrate an application in the area of urban reconstruction.

The scope of our contribution is limited to the analysis and synthesis of facade structures. We make no contribution to the problem of extracting facade layouts from images (we assume abstracted layouts as input) or the problem of facade texture synthesis (we show textured results for visualization purposes only and assume texture synthesis as a separate post-process).

2 Related Work

We review related work in three different areas: structural image inpainting, facade modeling, and facade analysis.

Structural Image Inpainting: Our work builds on the concept of image inpainting, especially algorithms that consider the structure of the observed part of the image. Sun et al. [2005] introduce an interactive system that synthesizes image patches by considering user-specified structural information in the unknown region. He and Sun [2012] propose an algorithm that is inspired by facade images. They assume that facade elements are repeated using only a few dominant offsets. After extracting these offsets, they use the offsets to generate a stack of shifted images and combine them via optimization. Korah et al. [2008] complete the partially occluded facade by modeling the grid patterns on facades as Near Regular Texture. They use an MRF model with MCMC optimization to discover the occluded regions. The most advanced algorithm is probably the recent method by Dai et al. [2013]. These models are a good starting point, but the most challenging problems remain unsolved: how to jointly optimize element positions and element attributes as well as completing structures with large unobserved regions?

Facade Modeling: During the last decade, various methods have been proposed to model facade layouts. One approach to encoding and generating facade layouts is to use grammars [Wonka et al. 2003; Müller et al. 2006]. The grammar rules encode shape replacement operations, typically splitting boxes (rectangles) into smaller boxes. While context free grammars are not as powerful for modeling, they can be nicely combined with other energy functions using rjMCMC [2011]. Grammars can also be used to parse facade layouts given an input image [Koutsourakis et al. 2009]. Another class of methods uses optimization techniques to generate facade layouts, e.g. Lin et al. [2011] and Bao et al. [2013]. While Lefebvre et al. [2010] work directly on pixels in an image, the method could also be used to synthesize layouts directly. Two recent methods [Yeh et al. 2013; Dai et al. 2013] generate new facade with a tile-based layouting algorithm. Yeh et al. [2013] propose a sampling algorithm for a probabilistic model using factor graphs, and Dai et al. [2013] propose a genetic algorithm to guide the consistency between synthesized content and the example.

Facade Analysis: Facade analysis is a building block of urban reconstruction and a recent survey [Musialski et al. 2013] reviews facade analysis in its broader context. One class of facade analysis methods works directly on images. The main goal of these methods is to segment a facade image into meaningful components. Müller et al. [2007] build a bridge between facade modeling and facade analysis by combining the procedural modeling pipeline of shape grammars with image analysis to derive facade models from images. Teboul et al. [2010] perform supervised learning using shape grammar priors to interpret building facades. The method has been extended using recursive split grammar and reinforcement learning [Teboul et al. 2013]. Martinovie et al. [2012] propose a three-layered approach for semantic segmentation of building facades. Shen et al. [2011] propose a method to automatically partition a facade in an adaptive manner, in which the splitting direction and the number and location of splitting planes are all adaptively determined. Musialski et al. [2012] propose a coherence-based interactive system to support non-local operations. Alhalawani et al. [2013] apply structure optimization to recover a semi-regular grid of facade elements. Since most of the facade elements are rectangles, a facade can be analyzed using low rank matrix decomposition [Ali et al. 2009; Yang et al. 2012]. Another class of analysis methods uses already segmented facade models as input. Three recent methods try to compute shape grammars from segmented facade layouts [Weissenberg et al. 2013; Martinovic and Van Gool 2013; Wu et al. 2014]. Alternatively, Zhang et al. [2013] propose a great facade representation based on layered grids that can also be used to analyze segmented facade layouts.

3 Overview

3.1 Overview

Using urban reconstruction as motivational example, we show how the core part of our work can be embedded in an application (See Fig. 1). The core part itself is the structure completion. Starting with a partial layout as input, we complete the layout using two main components:

- The first component is a statistical model that can take a facade layout as input and assign a probability score to measure how likely the layout is. The parameters of this model are learned from a database of complete layouts (Section 4).
- The second component is a planning algorithm that computes completed candidate structures. The candidate structures are then evaluated by the statistical model to either select the best one or to select a variety of possible completion results. The parameters of the completion algorithm can be learned using ideas from planning and reinforcement learning, i.e., policy optimization (Section 5).

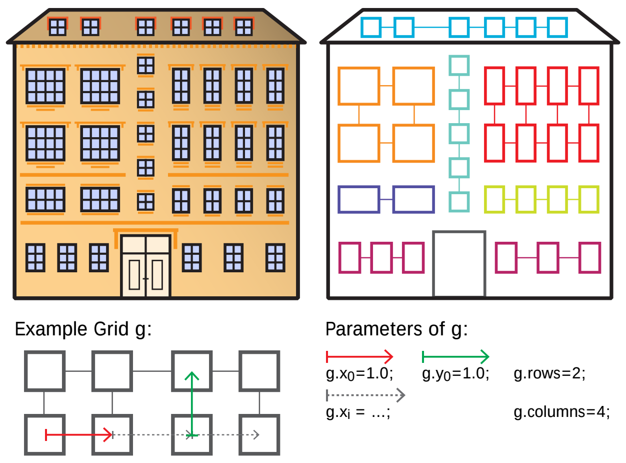

Figure 3: *Top: a layout consists of a set of elements, each element defined by a position, a bounding box, and appearance attributes. For facade layouts, the elements are typically organized by a set of grids. Bottom: the most important parameters of a single grid.*

3.2 Definitions

A layout is defined by a set of shapes $\{S_i\}$ inside a two-dimensional domain Ω. Ω is a polygon, typically a rectangle. A shape is defined by a location $Pos_i \in \mathbb{R}^2$ inside the domain that denotes the center of the shape. Further, the shape has a height h_i and width w_i. For non-rectangular shapes, height and width refer to the size of the bounding box. Each shape has a label l_i and appearance attributes, such as color, material, and depth (offset from the main facade plane). A grid layout is a layout where the shapes are organized by a set of grids (See Fig. 3 top). A similar structure was proposed by Zhang et al. [Zhang et al. 2013]. A grid g is defined by the dimensions of its bounding box ($g.width$ and $g.height$), the number of rows and columns in the grid ($g.rows$, $g.columns$), the position of the lower left corner in the domain ($g.position$), and a vector of spacing values for the rows and columns ($g.x_i$ and $g.y_j$) (See Fig. 3 bottom). In the context of a facade we also refer to the shapes as elements. We say that a facade F consists of a grid structure $G = \{g_i\}_{i=0}^{n-1}$, where g_i denotes the i^{th} grid and n is the number of grids. We denote the set of observed elements as $\{e_i\}$.

3.3 Facade Database

We generated a facade database of 100 facade images by combining our own images with the images collected from other authors [Zhang et al. 2013; Teboul et al. 2010; Musialski et al. 2012]. This database contains complete, unoccluded facades that are useful for learning facade structures and evaluating the algorithm. We use manual segmentation tools to process the image database, but automatic segmentation from previous work could also be used.

4 A Statistical Model for Grid Layouts

In this section, we define a set of attributes that contribute to good grid structures, and learn distributions over the attributes from a dataset of example facades. Then we use the learned distributions to score the quality of given grid structures. We classify scoring functions into three types, i.e., unary, binary, and global functions, and combine them into one unified statistical model using a factor graph. In the course of this project, we built and evaluated a large number of similar statistical models and we experimented with directed and undirected graphical models. The proposed model was designed to be simple enough so that it can be learned from a small database, but still expressive enough to handle challenging completion problems.

4.1 Unary Grid Functions

We first consider the scoring functions defined by the properties of one single grid.

Element Aspect Ratio: The size of an element is determined by its height and width. Instead of using a two parameters distribution we encode the element size as the aspect ratio. We build a Gaussian mixture model for each type of element e to encode the aspect ratio distribution as p_{as}.We estimate the number of Gaussian components automatically using the MDL criterion [Rissanen 1983]. The element types are defined by the different labels used in the element database. For an element e in grid g, we then use the scoring function:

$$f_{as}(g) = \ln p_{as}\left(\frac{e.height}{e.width}\right). \tag{1}$$

Element Spacing: In our analysis, we observed that the spacing between two elements in one grid is related to the size of the element, e.g., typically the spacing between two large elements is larger than the spacing between two small elements. We build two Gaussian mixture models to estimate the distribution of the ratio of element size and average spacing in x and y directions separately, i.e., $p_{ed}^H(\cdot)$ and $p_{ed}^V(\cdot)$. For an element e in grid g, we use the following scoring function:

$$f_{ed}(g) = \ln p_{ed}^H\left(\frac{e.width}{\overline{d}^H}\right) + \ln p_{ed}^V\left(\frac{e.height}{\overline{d}^V}\right), \tag{2}$$

where $\overline{d}^H$ and $\overline{d}^V$ are the average horizontal and vertical spacing between two elements in grid g respectively.

Grid Regularity: For each grid g, we define the grid regularity in x and y directions as $reg^H(g) = \frac{\sigma^H}{g.width}$ and $reg^V(g) = \frac{\sigma^V}{g.height}$ separately, where σ^H and σ^V are the standard deviations of the horizontal and the vertical spacing in grid g respectively. We analyze the grid regularity by estimating the distribution of reg^H and reg^V in the dataset. We define the following scoring function:

$$f_{gr}(g) = \ln p_{gr}^H\left(reg^H(g)\right) + \ln p_{gr}^V\left(reg^V(g)\right). \tag{3}$$

Grid Completeness: The grid completeness is a measure of how completely a grid is filled. It is computed by $comp(g) = \frac{\#elements}{g.rows \times g.columns}$, where $\#elements$ is the number of elements in grid g. We encode the distribution of grid completeness p_{gc} using a Gaussian mixture model learned from all the grids in the dataset. We define a scoring function based on grid completeness for grid g:

$$f_{gc}(g) = \ln p_{gc}\left(comp(g)\right). \tag{4}$$

4.2 Binary Grid Functions

While unary terms model the properties of one grid, binary terms capture relationships between two grids. We identified the pattern of grid arrangement and the alignment of elements, as the two important appearance properties of a facade.

Pattern of Interleaved Grids: To simplify the pattern problem, we only consider the patterns generated by two interleaved grids. We first analyze the pattern styles in the dataset. We define the

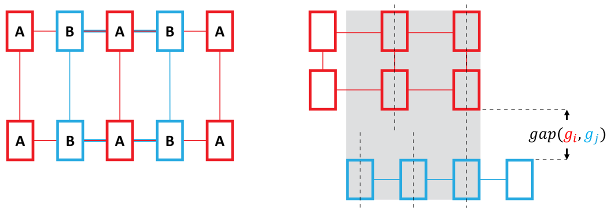

Figure 4: *Left: a pattern of two interleaved grids. Here, the pattern is AB because AB is the repeating sequence. Right: an example for the grid alignment computation, where* $ac^V(g_i, g_j) = \frac{2}{5}$.

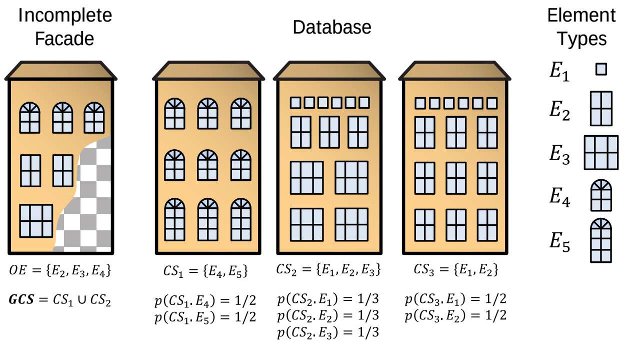

Figure 5: *An example of element compatibility. Left: an incomplete facade with observed element types E_2, E_3, and E_4. Middle: a facade database consisting of three facades. Right: The set $ElementList$ consisting of all elements in the database. At the bottom we show the computation of element probabilities and the computation of the GCS.*

pattern style as the minimal pattern presented by two interleaved grids, e.g., AB, ABA, AAB, AABB. We extract the pattern style of two interleaved grids by abstracting the element labels to A and B without considering the specific labels. See Fig. 4 left for an example. Then we list the patterns in the dataset, and create the histogram by counting the pattern style of each interleaved grid pair. The histogram is then normalized to form the discrete probability distribution. Given two interleaved grids g_i and g_j, we can define a scoring function as

$$f_{gp}(g_i, g_j) = \ln p_{gp}\left(\Theta(g_i, g_j)\right), \tag{5}$$

where $\Theta(\cdot,\cdot)$ is the pattern operator.

Grid Alignment: The problem of defining and learning distributions of alignment between two grids is difficult. However, there are a few ways to simplify the problem. First, we consider the alignment in the x and y direction separately. Second, we only consider alignment between grids where the projections of the grids overlap. For the vertical alignment problem, we first define the normalized vertical gap $\overline{gap}^V(g_i, g_j) = \frac{gap^V(g_i, g_j)}{F.height}$ between two grids. Then we define the alignment coefficient as $ac^V(g_i, g_j) = \frac{\#aligned\ coloumns}{\#overlapping\ coloumns}$. See Fig. 4 right for an example. We encode the vertical alignment distribution using a Gaussian mixture model. The horizontal alignment distribution is learned in the same manner. The alignment scoring function is defined as

$$f_{ga}(g_i, g_j) = \ln p^H_{ga}(ac^H|\overline{gap}^H) + \ln p^V_{ga}(ac^V|\overline{gap}^V). \tag{6}$$

4.3 Global Grid Functions

We define four scoring functions to evaluate global properties of the facade.

Element Compatibility: To enforce global consistency, we introduce a function to evaluate the compatibility of element types in the facade. First, we learn the element compatibility from example facades. We analyze the facade database to model a set of element types $ElementTypes = \{E_0, E_1, \ldots, E_{n-1}\}$. The most important goal is to find element types that work well together. We learn the joint occurrence of different elements by analyzing the facades and building coherence sets. Each facade contributes one coherence set. A *coherence set (CS)* is a set of all element types that occur together in a facade. Each element in a coherence set is assigned a probability. The probability of an element is estimated as the number of grids using the element divided by the total number of grids contributing to the coherence set. See Fig. 5 for an example.

In a second pass we merge coherence sets that contain exactly the same type of elements and update the probabilities accordingly. The probability of a coherence set itself is how many individual coherence sets were merged to build it. Given the set of all observed element types, e.g. $OE = \{E_2, E_3, E_4\}$, we try to find the *global consistency set (GCS)* that is a union of coherence sets $\{CS_i\}$ to cover all observed elements. We define the GCS as the smallest cover of OE using coherence sets:

$$\begin{aligned} GCS \quad &= \arg\min_{\cup CS_i} \# \left\{ \bigcup CS_i \right\}, \\ s.t. \qquad & OE \subseteq \bigcup CS_i. \end{aligned} \tag{7}$$

In practice we employ exhaustive search to find the GCS. If the optimization has multiple solutions, we sample according to the probability of the coherence sets to select one as GCS.

Given a set of grids $G = \{g_0, g_1, \ldots, g_{n-1}\}$, we evaluate the compatibility of element type by $f_{ec}(\cdot)$ based on the GCS defined above,

$$f_{ec}(G) = \ln\left(1 - \frac{\#GCS(G)}{\#OE(G)}\right), \tag{8}$$

where $\#GCS(G)$ is the number of the CSs in GCS covering G and $\#OE(G)$ is the number of element types in G. Note that $\frac{\#GCS(G)}{\#OE(G)} \in [\frac{1}{\#OE(G)}, 1]$ and $\frac{\#GCS(G)}{\#OE(G)} \to 1$ means that the compatibility of elements is weak, otherwise the compatibility is strong.

Grid Coverage: We define the grid coverage rate $cr(G) = \frac{A(\bigcup g_i)}{A(F)}$ to describe the percentage of covered area of the facade. In this equation $A(\bigcup g_i)$ is the area of the union of grid bounding boxes and $A(F)$ is the area of the facade. We learn the distribution of the grid coverage rate and define the following scoring function:

$$f_{gc}(G) = \ln p_{gc}(cr(G)). \tag{9}$$

Facade Border: For realistic results, it is important to consider both the spacing between elements and the spacing from elements to the facade boundary. Here we define the border width as the distance between the facade boundary and the boundary of the nearest element. We encode the distribution of the border width normalized by the average element size in the facade. We learn the distributions for the horizontal and the vertical border width separately. Given a set of grids G, we define a scoring function as

$$f_{fb}(G) = \ln p^H_{fb}(G) + \ln p^V_{fb}(G). \tag{10}$$

Facade Symmetry: We observed that the symmetry of a facade is typically related to the door position. Given example facades, we

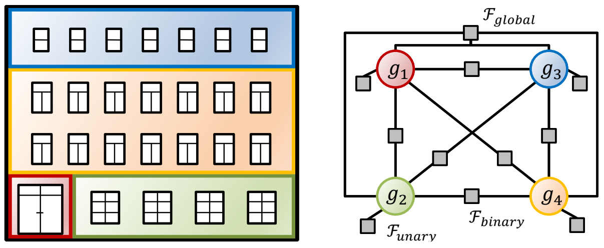

Figure 6: *Left: a grid structure of the facade. Right: a factor graph for the example on the left.Variable nodes are colored according to their corresponding grids on the left.*

compute the average door position and the area of the symmetric region. Then we discretize the values and build a 2D histogram of these two values. The histogram is then smoothed using kernel density estimation [Wand and Jones 1994]. We define a scoring function based on the learned distribution:

$$f_{fs}(G) = \ln p_{fs}(A(Sym(G))|v), \tag{11}$$

where v is the average door position and $Sym(G)$ is the symmetric region of grid structure G. If a door does not exist, we set v = 0.

4.4 Factor Graph Design

A factor graph is a probabilistic graphical model that decomposes a complex probability distribution over multiple variables into a set of smaller factors over subsets of the variables. In our problem, each grid $g_i, i = 0, 2, \dots, n-1$ determines a group of variables in the graphical model. The relationships between grids are encoded by the factors $\mathcal{F}$ derived from our scoring functions. Following common practice in the literature, each group of scoring functions is combined into a single factor. See Fig. 6 for a factor graph example. Note that the factor graph is dependent on the number of grids so that the factor graphs for different facades are usually different.

The unary factor combines the scoring functions evaluated for each grid by itself. It is connected to a single variable.

$$\mathcal{F}_{unary}(g_i) = \exp(w_{as}f_{as}(g_i) + w_{ed}f_{ed}(g_i) + w_{gr}f_{gr}(g_i) + w_{gc}f_{gc}(g_i)). \tag{12}$$

The binary factors connecting two grids are derived from the binary scoring functions.

$$\mathcal{F}_{binary}(g_i, g_j) = \exp(w_{gp}\sum f_{gp}(g_i, g_j) + w_{ga}\sum f_{ga}(g_i, g_j)). \tag{13}$$

The global factor connected to all grids in $G = \{g_0, g_1, \dots, g_{n-1}\}$ enforces global goodness of the facade

$$\mathcal{F}_{global}(G) = \exp(w_{ec}f_{ec}(G) + w_{gc}f_{gc}(G) + w_{fb}f_{fb}(G) + w_{fs}f_{fs}(G)). \tag{14}$$

The overall probability distribution resulting from the factor graph is defined as

$$p(G|\mathbf{w}) = \frac{1}{Z(F, \mathbf{w})} \prod_{\mathcal{F}} \mathcal{F}(Scope_{\mathcal{F}}(G)), \tag{15}$$

where $Scope_{\mathcal{F}}$ denotes the variables connected to factor $\mathcal{F}$ and $Z(F, \mathbf{w})$ is the facade-dependent partition function that normalizes the distribution. The distribution is parameterized by the vector of factor weights $\mathbf{w}$. These weights can be estimated using weight learning described next.

Weight Learning: The weight learning problem can be formulated as *maximum likelihood parameter estimation*. This weight estimation is an important building block when working with factor graphs. In computer graphics it was recently used in the context of coloring 2D graphics [Lin et al. 2013].

Equation 15 can be reformulated to group the product according to the different weights in the weight vector $\mathbf{w}$.

$$p(G|F, \mathbf{w}) = \frac{1}{Z(F, \mathbf{w})} \prod_{w \in \mathbf{w}} \exp(w \cdot f_w(G, F)), \tag{16}$$

where $f_w(G)$ is the sum of all the scoring functions f that share the weight $w \in \mathbf{w}$.

The log-likelihood of the weights given a dataset $\mathcal{D}$ is

$$\mathcal{L}(\mathbf{w}|\mathcal{D}) = \sum_{(F,G) \in \mathcal{D}} \sum_{w \in \mathbf{w}} w \cdot f_w(G, F) - \ln Z(\mathbf{w}). \tag{17}$$

Then, the optimal weight vector is computed as

$$\mathbf{w*} = \arg\max_{\mathbf{w}} \mathcal{L}(\mathbf{w}|\mathcal{D}). \tag{18}$$

This problem can be solved using gradient ascent. The partial derivatives with respect to the weights are

$$\nabla_w \mathcal{L}(\mathbf{w}|\mathcal{D}) = \sum_{(F,G) \in \mathcal{D}} f_w(G, F) - \mathbb{E}_{\mathbf{w}}[f_w(G, F)], \tag{19}$$

where $\mathbb{E}_{\mathbf{w}}$ is the expectation under the model with weights $\mathbf{w}$. Since the expectation term is extremely expensive to compute, we apply *Contrastive Divergence*(CD) [Hinton 2002] to approximate the likelihood gradient

$$CD_w^k(\mathbf{w}|\mathcal{D}) = \sum_{(F,G) \in \mathcal{D}} f_w(G, F) - f_w(\hat{G}, F), \tag{20}$$

where $\hat{G}$ is a grid structure generated by the completion algorithm described in the next section. We use a grid structure in which 10% of the elements were randomly removed from G as input.

5 Structure Candidate Generation

In this section we present a method to complete an incomplete layout. This problem is challenging, because only layouts with fairly regular spacing and proper alignment are reasonable. In our experiments, we observed that it is very difficulty to find local operations that can convert a reasonable layout into another reasonable layout. This is especially difficult, because the observed elements have to be respected as hard constraints. Further, a straightforward randomized search algorithm such as simulated annealing fails completely if only simple add, delete, and move operations are employed (See Fig. 14). Therefore, instead of using MCMC to sample from the statistical model directly, we propose to draw from the planning and reinforcement learning literature to design an algorithm that mainly generates reasonable completion candidates (See [LaValle 2006] for an introduction to planning algorithms). We cast the problem as an optimization problem to find a completion that maximizes the layout score computed by Equation 15. As our algorithm generates multiple variations during its execution, we can sort these variations according to their score and either select the best one or use multiple solutions of different ranks to obtain different suggestions to

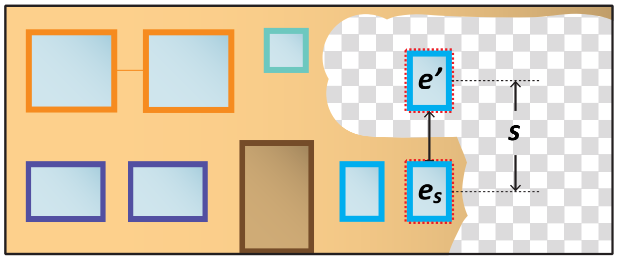

Figure 7: *A policy computes an action given the current state. The action consists of selecting a seed element e_s and an extension element e' with spacing s.*

complete a given layout. A limitation of this approach is that the sampling of the statistical model is biased.

Our algorithm searches a state space S starting from a starting state $s_0 \in S$ (the observed partial layout). Each state corresponds to a layout. In a state s_i, we need to select an action a_i from a possibly (infinite) number of available actions A. Based on a heuristic we can classify the layouts as complete or incomplete. A state s corresponding to a complete layout is considered to be a goal state. In a goal state s, the algorithm receives a reward $R(s)$ computed by Equation 15. We can now define the value of a state using the Bellman equation:

$$V(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} T(s,a,s')V(s'). \tag{21}$$

The transition model $T(s,a,s')$ defines the transition probabilities of ending up in state s' when selecting action a from state s. In our case, the transition model is deterministic given an action a and we use $\gamma = 1$ for the discount factor. A policy π is a function that maps a given state s to an action $a = \pi(s)$ and we would like to compute an optimal policy:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} T(s,a,s')V(s'). \tag{22}$$

Our first approach was to use feature extraction from states to learn approximate values $V(f(s))$ by backpropagating the rewards to intermediate states (where $f(s)$ is the function that extracts features from states). Unfortunately, this proved to be quite challenging, so we opted for a more direct approach using policy optimization. Our proposed solution is to design a probabilistic policy that can be configured by a parameter vector $\boldsymbol{\lambda}$ which is explained later.

An important component of our algorithm is that the policy π mainly generates actions that lead to meaningful layouts. To keep the algorithm simple, the policy only considers actions that add one facade element to the current layout. We do not use actions that move or delete elements. Therefore, we generate completed structure candidates by adding elements successively to expand the observed region.

5.1 Policy Design

A policy has a set of parameters and it generates an action based on the following steps. First, we select a seed element e_s. Second, we decide the extension direction. Third, we compute the extension spacing s, and finally we decide the label of the new element e'. See Fig. 7 for an example. The vector λ encodes the parameters of the policy.

Seed Element Selection: We use two methods for selecting a seed element e_s. The first method selects a seed element randomly among all candidates. The second method selects a seed element based on a rank score. The parameter λ_0 is used to define the probability of selecting the first method (the probability for selecting the second method is then $1 - \lambda_0$). Given a potential seed element e_s, we define its rank as

$$r(e_s) = \lambda_1 \cdot \delta_c(e_s) + (1 - \lambda_1) \cdot \left(1 - \frac{A(g)}{A(F)}\right). \tag{23}$$

The first term computes the concavity of e_s and favors seed elements in concave regions. The second term favors seed elements in small grids to be selected first. The parameter λ_1 is a weight to trade off the two different terms. $\delta_c(e_s) = 1$ if e_s is inside of the bounding box of its grid and 0 if it is on the boundary. g is the grid that e_s belongs to and $A(\cdot)$ computes area.

Extension Direction: We use two methods to select the extension direction. The first method selects the direction randomly. The second method selects the direction according to fixed priorities. The parameter λ_2 specifies the probability of selecting the first method. For the second method, the parameter λ_3 encodes a permutation of possible expansion directions to indicate their priority. We allow an action to extend e_s in five directions, i.e., left, right, up, down, and *concave fill*. The direction concave fill tries to extend in such a way that the bounding box of the grid does not change.

Extension Label: We use two methods to select a label for the extension element e'. The first method copies the label from e_s. The second method itself chooses from three options: sampling the label from the observed labels, from the database, or set the label to be NULL (which creates some empty space). We use the parameter λ_4 to encode the probability of selecting the first method and the parameters λ_5, λ_6, and $1 - \lambda_6 - \lambda_5$ to specify the probabilities for the three options in the second method.

Extension Spacing: We use three methods to select the spacing s between the seed element e_s and the extension element e'. The first method copies the the spacing that the element e_s has with its neighbor from the opposite direction. The second method samples the spacing from the observed spacings. The third method samples the spacing from observed spacings and adds a random offset. We set the parameters λ_7 and λ_8 to encode the probabilities for selecting between these three methods.

Other Parameters: We also use two additional modifications. A newly placed element can be snapped to the position of other elements already placed on the facade (with probability λ_9). Further, we use an operation that copies the seed element symmetrically to the other side of the facade with probability λ_{10}.

Based on the parameters $\boldsymbol{\lambda} = \{\lambda_0, \lambda_1, \ldots, \lambda_{10}\}$ a policy π can select an action a_i for each state s_i.

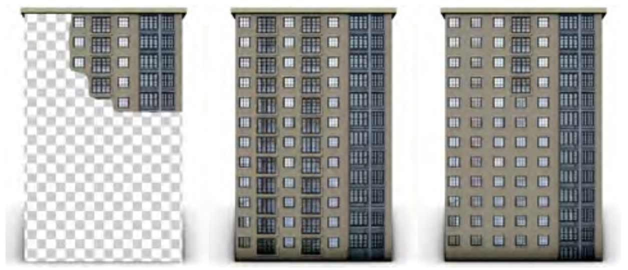

Figure 8: *Effect of policy optimization. From left to right: incomplete facade, a completion result using policy optimization, and a completion result with a fixed specified policy.*

5.2 Policy Optimization

One important problem is how to set the parameters $\boldsymbol{\lambda}$ for policy π. It is difficult for users to set $\boldsymbol{\lambda}$ directly, because it is hard to find a single policy that works with all facades. We therefore formulate the problem as policy optimization. For each facade, we compute the optimal parameters by solving the following optimization problem:

$$\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}} \sum_{s' \in S} T(s, \pi(s, \boldsymbol{\lambda}), s')V(s'), \tag{24}$$

where $\pi(s, \boldsymbol{\lambda})$ is the policy with parameters $\boldsymbol{\lambda}$.

We employ a genetic algorithm to solve the problem. We set the population size to 10. A crossover is computed by randomly selecting a parent for each parameter λ_i. A mutation is computed by randomly selecting a parameter $\lambda_i \in \boldsymbol{\lambda}$ and then randomly mutating by sampling an offset from a Gaussian distribution. As initialization, we first learn optimal policies for facades in the database and select the 10 most different ones. We use these 10 policies as initial policies for each facade. Figure 8 shows two completion results of the same incomplete facade, one facade is completed with a fixed user specified policy and the other one is completed using policy optimization. The policy optimization correctly interprets the column structure and can also generate an interleaved pattern. The result of the fixed policy is not plausible.

6 Results

We implemented the framework in C++ and MATLAB. For our evaluation we used a computer with two 2.53Ghz Intel core i5 processors and 4GB main memory.

Qualitative Analysis: We selected a set of facades not used as training data in the database for all results shown in the paper. While there is some existing work on structure completion, it cannot really be applied to our problem for two reasons. First, we compute structure completion from abstracted layouts and not from images. Second, previous work is only able to complete smaller missing regions. We believe that the most promising competitor of our work is a recent paper by Dai et al. [Dai et al. 2013]. They kindly agreed to send us completed versions based on our image data or images of abstracted layouts. However, they reported that they would need additional extensions to handle this type of data. In our observation, this competing algorithm is good at completing a single grid of facade elements, when each row and column has at least one (but better two) observed elements. Therefore, we opted to perform an informal user study where we show users our completion result and the ground truth (in randomized order). Then we ask the users to select which facade layout appears more plausible (reasonable). The user can select one of the layouts or choose the option that both are equally plausible. We had 134 users participate in an online study. In total, among all users the votes were distributed as follows. Ground truth data received 31.5%, our completion received 40.2%, and both equally received 28.3% of all votes. In Fig. 9 we show the samples used in the user study and the breakdown of votes per completion result. Interestingly, we can observe a slight preference for our completions and also some facades where either our completion or the ground truth was strongly preferred. Based on this user study and our own visual analysis of the results, we conclude that most of our results have very high quality.

There are two main ways to categorize the input to a structure completion problem: completeness and coherence. Completeness corresponds to how much information is missing and coherence measures how spread out the observed elements are. In the user study we tried to mix examples of coherent and incoherent input as well

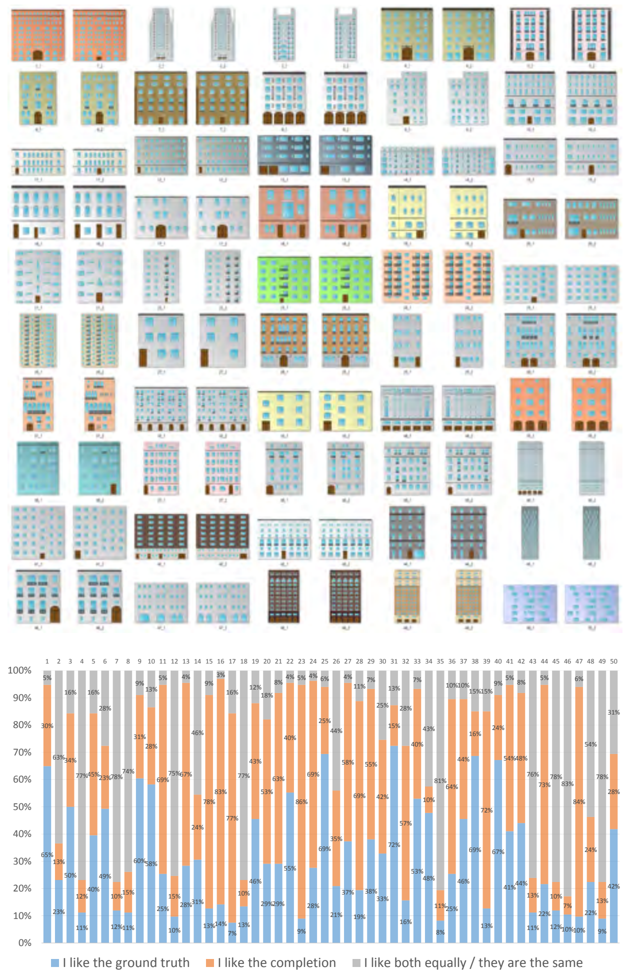

Figure 9: *Top: thumbnails of all 50 examples used in our user study (see Sec. 6, and additional materials), each image pair shows the ground truth (left) and our completion (right). We asked 126 persons in an online study to decide which of the two possible completions appears more plausible. Bottom: results of the study. During the study the user does not know which of the two facades is the ground truth.*

as vary the amount of structure that is missing. We selected two example facades to show how the completion changes when more and more information is available (See Fig. 10). As expected, the result is more similar to the ground truth layout when more elements are observed. However, all completions seem plausible. Fig. 11 illustrates two examples of incoherent inputs used in the user study.

Our algorithm computes multiple variations for each completion example that are ranked by the proposed statistical model. In the user study we always use the highest ranked completion. In Fig. 12 we illustrate multiple different completions for the same facade.

Limitations: Our statistical model typically ranks regular facades higher. It is therefore not possible to match the ground truth if part of an irregular facade is observed (See Fig. 13). Further, we can automatically select the 3D models for facade elements from a database. However, we did not implement an automatic solution to extract the wall texture from an input image. Therefore, the wall textures in all realistic renderings are generated semi-automatically using Photoshop. Only the abstracted layouts shown in the user study can be generated automatically. In future work, we would like

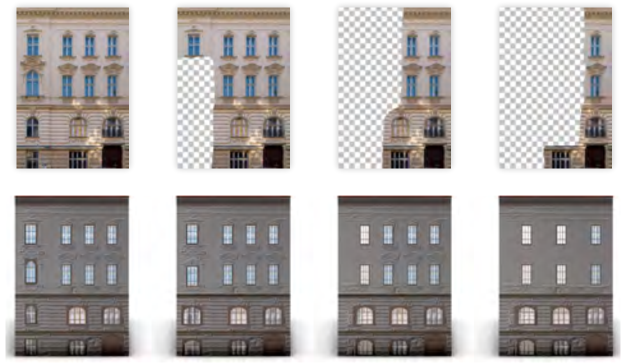
Figure 10: *Three examples how the number of observed elements influences the completion results. Left: input layout. Middle to Right: completions for a decreasing number of observed elements. The bottom row is a stylized visualization.*

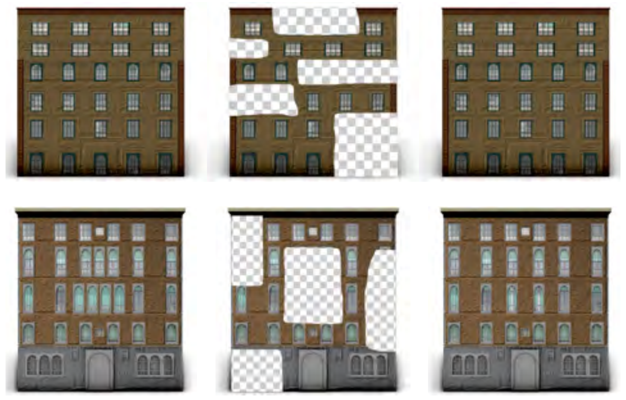
Figure 11: *Two examples of the completion results for observed elements in disconnected regions. From left to right: ground truth, incomplete facade, and completion results.*

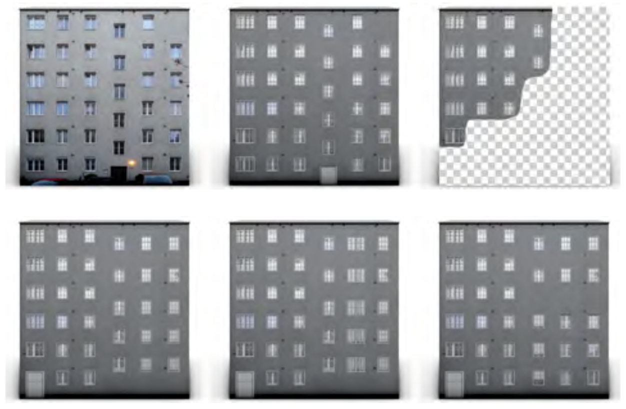
Figure 12: *Given the same incomplete facade, our algorithm can generate multiple different completions. Top: ground truth, stylized visualization and observed elements. Bottom: three possible completions.*

to evaluate if existing work can be used to solve this problem, or if a new texture synthesis method is required.

Comparison to Simulated Annealing: We demonstrate some aspects of the difficulty of the completion problem by comparing to a straightforward implementation of simulated annealing. We use the following operations: add an element at a random position, delete a random element, move an element along a randomly sampled offset vector (drawn from a Gaussian distribution). Despite our effort in tuning the parameters (probabilities) and a long running time, the algorithm is completely unable to find meaningful completion results (See Fig. 14).

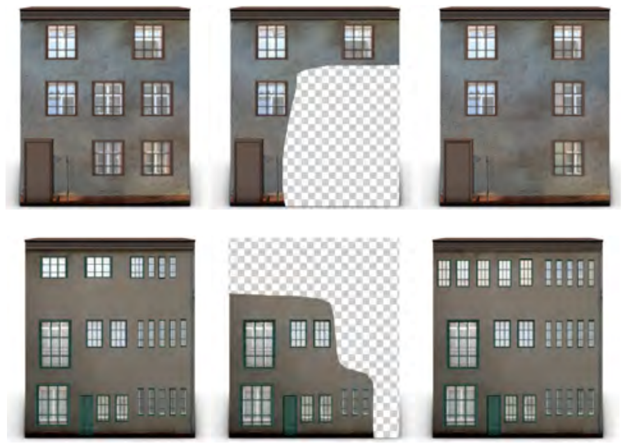
Figure 13: *Limitations of our algorithm. In these examples the ground truth is shown left, the observed elements in the middle, and our completion on the right. Since our completion favors regularity, the highest ranked completion does not match the ground truth in irregular facades.*

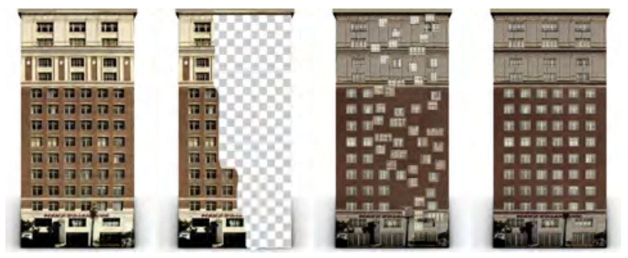
Figure 14: *A straightforward implementation of simulated annealing is not able to generate meaningful completions, because the algorithm is not able to iteratively improve the layouts to become more plausible. From left to right: ground truth, observations, simulated annealing, our completion.*

Timing: For all examples used in the user study, we generate 10,000 structure candidates during policy optimization for each incomplete facade and select the best one. We timed the computation times for all examples. The average time for the complete algorithm is 16.7 seconds. The timing depends on how many elements are missing and the complexity of the facade layout. But since we generate the structure candidates using simple actions, the variance of computation time is not high. Parameter learning of the statistical model takes 44.1 seconds for the weights of the factor graph.

Training Set Size: We perform a stress test to estimate the necessary size of the training dataset for our statistical model. At the beginning we learn a weight of each scoring function in the statistical model with the original training dataset (100 facades). Then we gradually reduce the size of it to analyze the changes of the weights. At each time we randomly exclude 10 facades from the dataset and learn the weights. We repeat the process three times for each case. Fig. 16 reports the average and the standard deviation of

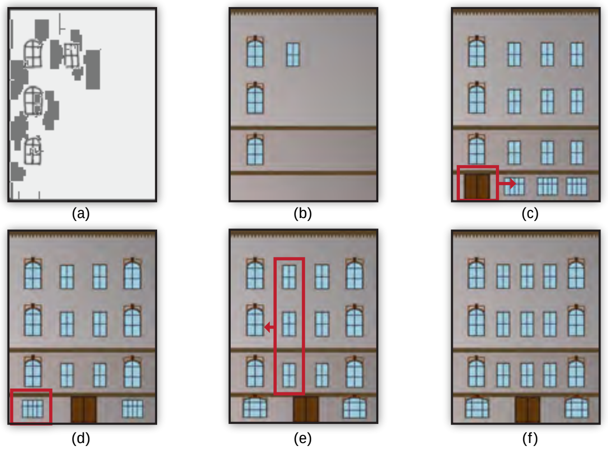

Figure 15: *A facade sketching application. The user sketches four elements (a) and the system selects suitable elements from a database (b) and suggests a completion (c). The user moves the door to a new position resulting in completion (d). Then, the user changes an element type leading to facade (e). Finally, the user moves a column of windows so that more windows are generated in the final result (f).*

each weight. We believe that this test shows that our model can be learned with a comparatively small number of training layouts. The main reason is that the model itself is not very complex and has a low number of parameters.

Scoring Functions: We complete a facade with parts of the model disabled to demonstrate the effect of each term in the statistical model. In Fig. 17 we show the effect of the facade symmetry term. The effects of the other terms are shown in Fig. 18. We can see that each of the terms contributes to the final result and that leaving out any term reduces the overall quality.

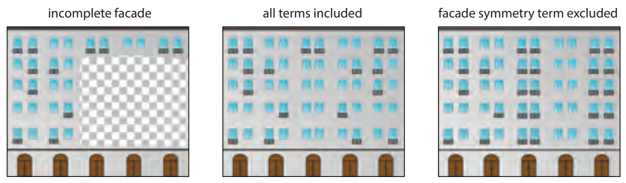

Figure 17: *The effect of the facade symmetry scoring function. From left to right: incomplete facade, a completion with all the scoring functions, and completion without symmetry function.*

Applications: Our structure completion can be used in multiple applications. One application is for urban reconstruction where a user can fit 3D mass models to buildings and complete facades to generate 3D models of partially occluded buildings (See Figs. 1, 20, and 21). The facade structure and the 3D model for doors and windows are placed automatically. Another example application is a structure suggestion module for a facade modeling interface. A user can sketch facade elements on a facade and ask the system to complete a design. Then, the completed result can be further edited and new completions can be generated. We show such an editing sequence in Fig. 15. The second application has not been fully implemented, but the completions are computed with our algorithm. In future work, we would like to explore the integration of our structure completion framework to multiple applications in more detail.

Alternative Approaches: One idea to complete a missing facade would be to adapt the idea of low rank matrix completion. As other authors noted before, low rank matrices are a good model for facade structures, e.g. [Yang et al. 2012]. However, low-rank matrix completion cannot be applied directly, because it requires that each row or column has at least some observations. Most of the examples shown in this paper do not fulfill this criteria. An illustration why low rank completion does not work, is given in Fig. 19. While the result is a rectangular pattern, there is no control over window spacing and window size. To extend the low rank matrix idea to layout completion, one possible approach could be to add additional priors on the size and spacing of the patterns encoded in the basis vectors. We believe that this is an interesting problem for future work in low rank matrix completion.

Another idea to approach the facade completion problem is to derive a grammar consistent with the observations. This idea was used in facade reconstruction to make a grammar consistent with an image [Koutsourakis et al. 2009]. A general problem with grammars is that it is difficult to write a single grammar that can capture the complete design space of all possible facades. Since we are interested in more complex structures, we conjecture that only some of our examples could be generated with existing shape grammars. Current grammars are only suitable to encode a small class of facade layouts, e.g. Haussmannian facades. We consider the idea of grammar-based completion another promising approach to solve the layout completion problem. One challenge to tackle would be the generation of better shape grammars to encode a wider class of facade designs.

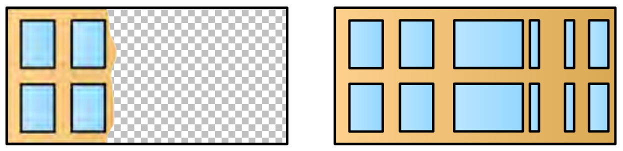

Figure 19: *Left: observed elements. Right: low rank completion. There is no control over window spacing and window size.*

7 Conclusions

We proposed a framework for structure completion for facade layouts. In contrast to previous work we pose the question for partially observed and segmented facades. While the pre-segmentation makes the problem easier, we solve a significantly more difficult problem. The first challenge of this work is to jointly estimate the position as well as the attributes of missing elements. The second challenge is to complete structures with only a few observations by learning plausible completions from a database. We tackle these challenges and present the first solution to this problem. We use a statistical model that can help to evaluate layouts and a planning algorithm that generates candidate layouts.

Acknowledgement

We thank the anonymous reviewers for their constructive comments. This research was partially funded by the Visual Computing Center of King Abdullah University of Science and Technology (KAUST), the National Natural Science Foundation of China (No. 61222206), the One Hundred Talent Project of the Chinese Academy of Sciences, the U.S. National Science Foundation (CCF 0643822), and the Austrian Science Funds (FWF P24600-N23).

References

AlHalawani, S., Yang, Y.-L., Liu, H., and Mitra, N. J. 2013. Interactive facades analysis and synthesis of semi-regular

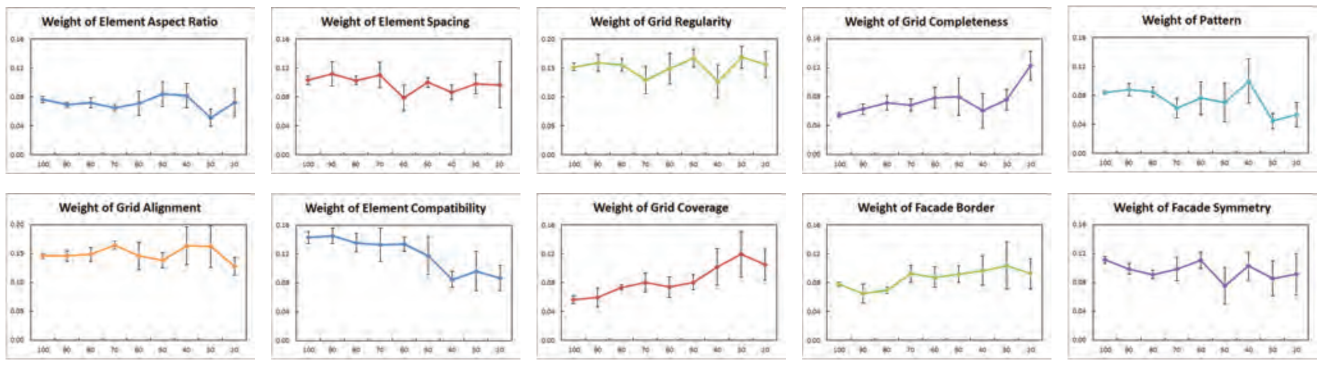

Figure 16: *Statistics of weight values on the different sizes of dataset. The average of each weight changes significantly and the standard deviation gets larger when the size of training dataset is smaller than* 70.

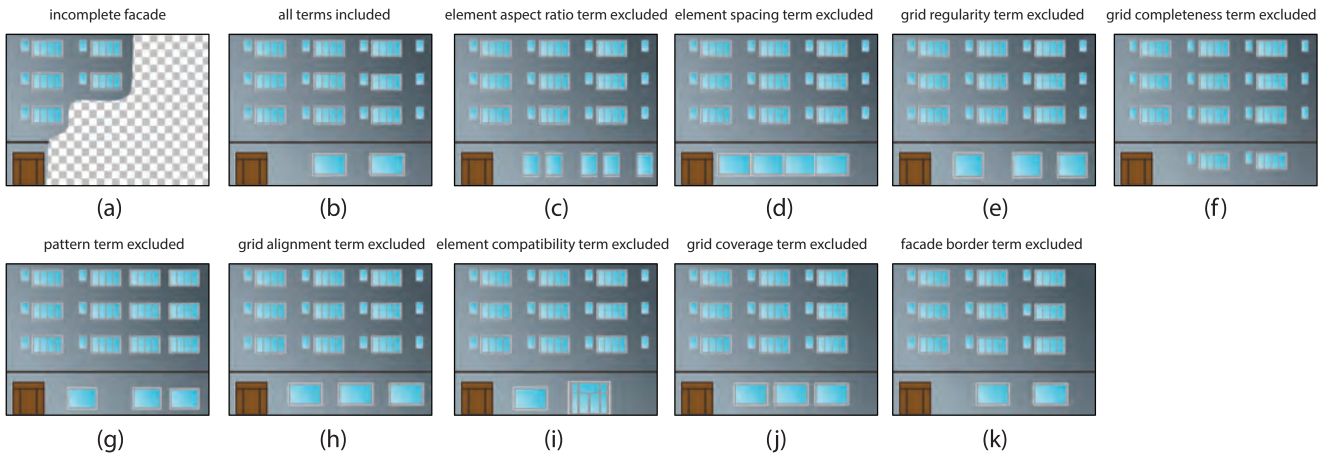
incomplete facade
all terms included
element aspect ratio term excluded
element spacing term excluded
grid regularity term excluded
grid completeness term excluded
(a)
(b)
(c)
(d)
(e)
(f)
pattern term excluded
grid alignment term excluded
element compatibility term excluded
grid coverage term excluded
facade border term excluded
(g)
(h)
(i)
(j)
(k)

Figure 18: *The effect of each scoring function. (a) Incomplete facade. (b) A completion result generated with all the scoring functions. (c-k) Ablation of individual scoring functions and its effect on the completion.*

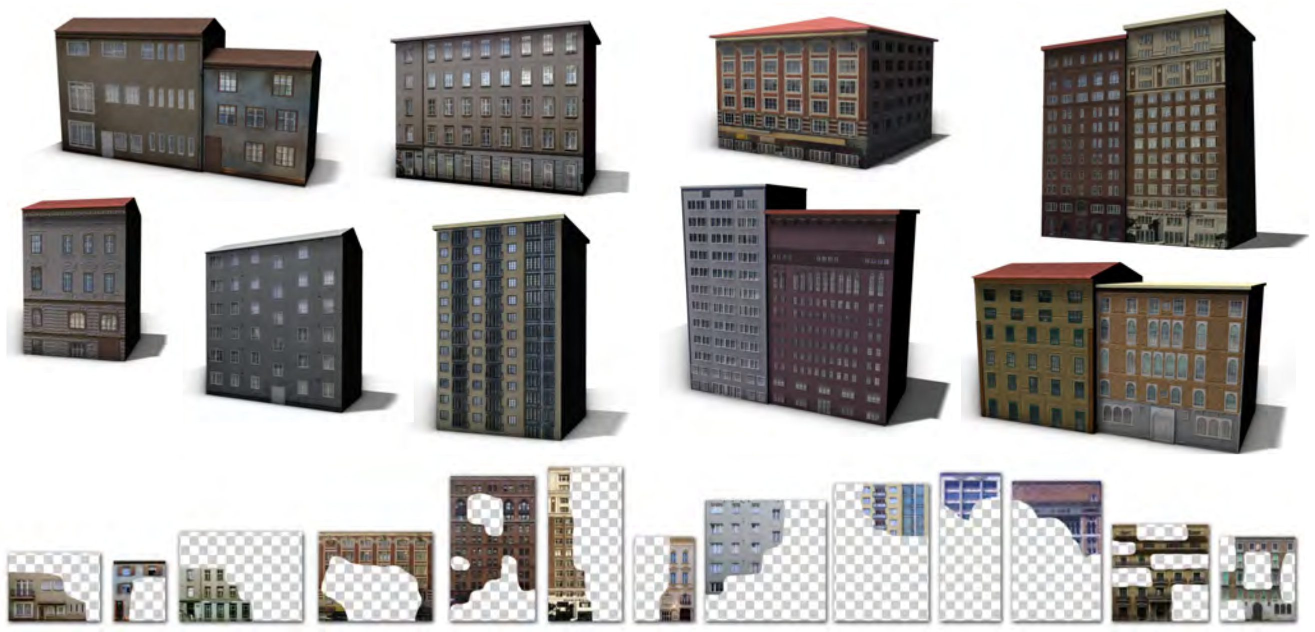

Figure 21: *Several 3D models created using completion results.*

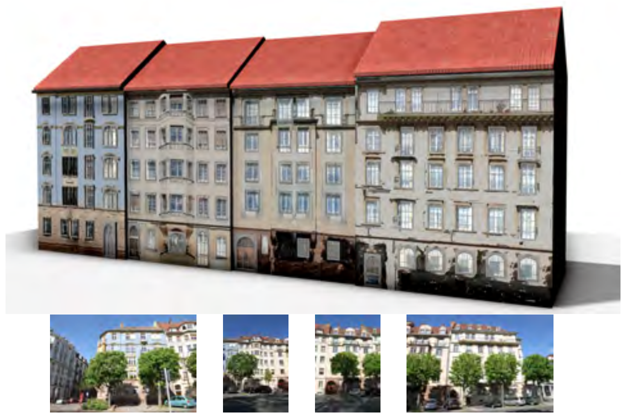

Figure 20: *Completion of a series of buildings along a street occluded by trees.*

facades. *Computer Graphics Forum 32*, 2, 215–224.

ALI, S., YE, J., RAZDAN, A., AND WONKA, P. 2009. Compressed facade displacement maps. *IEEE Trans. on Vis. and Comp. Graph 15*, 2, 262–273.

BAO, F., SCHWARZ, M., AND WONKA, P. 2013. Procedural facade variations from a single layout. *ACM Trans. Graph. 32*, 1 (Jan.), 8:1–8:13.

BERTALMÍO, M., SAPIRO, G., CASELLES, V., AND BALLESTER, C. 2000. Image inpainting. In *Proc. of SIGGRAPH 2000*, 417–424.

DAI, D., RIEMENSCHNEIDER, H., SCHMITT, G., AND VAN GOOL, L. 2013. Example-based facade texture synthesis. In *ICCV*, 1065–1072.

HE, K., AND SUN, J. 2012. Statistics of patch offsets for image completion. In *ECCV*, 16–29.

HINTON, G. E. 2002. Training products of experts by minimizing contrastive divergence. *Neural Computation 14*, 8, 1771–1800.

KORAH, T., AND RASMUSSEN, C. 2008. Analysis of building textures for reconstructing partially occluded facades. In *ECCV*, 359–372.

KOUTSOURAKIS, P., SIMON, L., TEBOUL, O., TZIRITAS, G., AND PARAGIOS, N. 2009. Single view reconstruction using shape grammars for urban environments. In *ICCV*, 1795–1802.

LAVALLE, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

LEFEBVRE, S., HORNUS, S., AND LASRAM, A. 2010. By-example synthesis of architectural textures. *ACM Trans. Graph. 29*, 4 (July), 84:1–84:8.

LIN, J., COHEN-OR, D., ZHANG, H., LIANG, C., SHARF, A., DEUSSEN, O., AND CHEN, B. 2011. Structure-preserving retargeting of irregular 3D architecture. *ACM Trans. Graph. 30*, 6, 183:1–183:10.

LIN, S., RITCHIE, D., FISHER, M., AND HANRAHAN, P. 2013. Probabilistic color-by-numbers: Suggesting pattern colorizations using factor graphs. *ACM Trans. Graph. 32*, 4 (July), 37:1–37:12.

MARTINOVIC, A., AND VAN GOOL, L. 2013. Bayesian grammar learning for inverse procedural modeling. In *CVPR*, 201–208.

MARTINOVIĆ, A., MATHIAS, M., WEISSENBERG, J., AND VAN GOOL, L. 2012. A three-layered approach to facade parsing. In *ECCV*, 416–429.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND GOOL, L. V. 2006. Procedural modeling of buildings. *ACM Trans. Graph. 25*, 3 (July), 614–623.

MÜLLER, P., ZENG, G., WONKA, P., AND GOOL, L. V. 2007. Image-based procedural modeling of facades. *ACM Trans. Graph. 26*, 3 (July), 85:1–85:9.

MUSIALSKI, P., WIMMER, M., AND WONKA, P. 2012. Interactive coherence-based façade modeling. *Computer Graphics Forum 31*, 2, 661–670.

MUSIALSKI, P., WONKA, P., ALIAGA, D. G., WIMMER, M., VAN GOOL, L., AND PURGATHOFER, W. 2013. A Survey of Urban Reconstruction. *Computer Graphics Forum 32*, 6, 146–177.

RISSANEN, J. 1983. A universal prior for integers and estimation by minimum description length. *The Annals of statistics*, 416–431.

SHEN, C.-H., HUANG, S.-S., FU, H., AND HU, S.-M. 2011. Adaptive partitioning of urban facades. *ACM Trans. Graph. 30*, 6 (Dec.), 184:1–184:10.

SUN, J., YUAN, L., JIA, J., AND SHUM, H.-Y. 2005. Image completion with structure propagation. *ACM Trans. Graph. 24*, 3 (July), 861–868.

TALTON, J. O., LOU, Y., LESSER, S., DUKE, J., MĚCH, R., AND KOLTUN, V. 2011. Metropolis procedural modeling. *ACM Trans. Graph. 30*, 2 (Apr.), 11:1–11:14.

TEBOUL, O., SIMON, L., KOUTSOURAKIS, P., AND PARAGIOS, N. 2010. Segmentation of building facades using procedural shape priors. 3105–3112.

TEBOUL, O., KOKKINOS, I., SIMON, L., KOUTSOURAKIS, P., AND PARAGIOS, N. 2013. Parsing facades with shape grammars and reinforcement learning. *IEEE Trans. Pattern Anal. Mach. Intell 35*, 7, 1744–56.

WAND, M. P., AND JONES, M. C. 1994. *Kernel Smoothing*. Crc Press.

WEISSENBERG, J., RIEMENSCHNEIDER, H., PRASAD, M., AND VAN GOOL, L. 2013. Is there a procedural logic to architecture? In *CVPR*, 185–192.

WONKA, P., WIMMER, M., SILLION, F. X., AND RIBARSKY, W. 2003. Instant architecture. *ACM Trans. Graph. 22*, 3 (July), 669–677.

WU, F., YAN, D.-M., DONG, W., ZHANG, X., AND WONKA, P. 2014. Inverse procedural modeling of facade layouts. *ACM Trans. Graph. 33*, 4 (July), 121:1–121:10.

YANG, C., HAN, T., QUAN, L., AND TAI, C.-L. 2012. Parsing façade with rank-one approximation. 1720–1727.

YEH, Y.-T., BREEDEN, K., YANG, L., FISHER, M., AND HANRAHAN, P. 2013. Synthesis of tiled patterns using factor graphs. *ACM Trans. Graph. 32*, 1 (Jan.), 3:1–3:13.

ZHANG, H., XU, K., JIANG, W., LIN, J., COHEN-OR, D., AND CHEN, B. 2013. Layered analysis of irregular facades via symmetry maximization. *ACM Trans. Graph. 32*, 4 (July), 121:1–121:10.