Automatic Constraint Detection for 2D Layout Regularization

Haiyong Jiang, Liangliang Nan, Dong-Ming Yan, Weiming Dong, Xiaopeng Zhang, Peter Wonka

Abstract—In this paper, we address the problem of constraint detection for layout regularization. The layout we consider is a set of two-dimensional elements where each element is represented by its bounding box. Layout regularization is important in digitizing plans or images, such as floor plans and facade images, and in the improvement of user-created contents, such as architectural drawings and slide layouts. To regularize a layout, we aim to improve the input by detecting and subsequently enforcing alignment, size, and distance constraints between layout elements. Similar to previous work, we formulate layout regularization as a quadratic programming problem. In addition, we propose a novel optimization algorithm that automatically detects constraints. We evaluate the proposed framework using a variety of input layouts from different applications. Our results demonstrate that our method has superior performance to the state of the art.

Index Terms—layout regularization, constraint detection, constraint analysis, linear integer programming.

1 Introduction

We propose an algorithm for the regularization of layouts. In this paper, a layout refers to a two-dimensional arrangement of objects. Layouts arise in a variety of applications. For example, they can come from digitized architectural floor plans, digitized facade images, image and text layouts on slides, line drawings, and graph drawings. In practice, when a layout is designed or digitized from another source (e.g., images), it is inevitable that noise will occur via imprecise user input. Elements in an idealized layout exhibit some regularities, e.g., they are aligned, of the same size, or uniformly distributed along a specific direction. However, in the aforementioned applications, these regularities typically disappear due to approximate user input. In this work, we seek to detect and restore these regularities by eliminating the noise that occurs during the layout design or digitization stage.

We offer three reasons for why this is an important problem. First, high-level shape analysis is a popular topic in computer graphics. Many available methods rely on correctly extracted relationships to analyze a scene [1]. Even if the input and output of our regularization look similar, it is important that correct relationships are extracted. Our motivation for this paper is to build datasets for machine learning techniques for layout synthesis. Second, a regularized layout compresses better than a noisy one. Regularization is thus important to the efficient representation of layouts. Third, in most cases, the visual differences are noticeable and the regularized layout looks better than the original one.

Regularization of layouts is challenging because of the complexity of constraints. Here discuss a few such constraints: 1) Elements can be partially aligned (e.g., elements are bottom aligned, but not top aligned). 2) Large elements can be aligned with multiple objects (e.g., top aligned with one and bottom aligned with another). 3) Elements can be missing from a regular pattern. 4) The spacing between rows and/or columns can be irregular. Fig. 1 shows the complexity of possible constraints in an example layout.

A key ingredient in regularization is the design of the layout model. A simple layout model has only a few parameters and therefore the fitting process is fairly robust. These simple models, e.g., a set of regular grids, are popular for automatic pattern analysis in images [2] and three-dimensional (3D) shapes [3], [4]. Unfortunately, this simple data model is limited in its applicability to a large class of layouts, e.g., the layout shown in Fig. 1. A complex model typically has many parameters and can fit a large number of layouts. However, such a model is not very robust to noise.

An initial framework for regularization was presented by Pavlidis and Van Wyk [5]. They propose a simple greedy algorithm to detect constraints from a layout. By constrast, we use an optimization approach based on four steps. First, we extract constraint candidates. Second, we score the likelihood of constraints based on energy functions. Third, we use global optimization using linear integer programming to select a subset of the constraint candidates that

- *H. Jiang is with the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and KAUST, Thuwal 23955-6900, Saudi Arabia. Email: haiyong.jiang@nlpr.ia.ac.cn.*
- *L. Nan is with KAUST, Thuwal 23955-6900, Saudi Arabia. Email: liangliang.nan@gmail.com*
- *D.-M. Yan is with KAUST, Thuwal 23955-6900, Saudi Arabia, and the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China. Email: yandongming@gmail.com. D.-M. Yan is the corresponding author.*
- *W. Dong and X. Zhang are with the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China. Email: weiming.dong@ia.ac.cn, xiaopeng.zhang@ia.ac.cn.*
- *P. Wonka is with KAUST, Thuwal 23955-6900, Saudi Arabia, and Arizona State University, Tempe, AZ 85287-8809. E-mail: pwonka@gmail.com.*

Manuscript received 13 Feb. 2015; revised 1 Sept. 2015; accepted 6 Sept. 2015. Date of publication ; date of current version.
Recommended for acceptance by S.-M. Hu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2015.2480059.

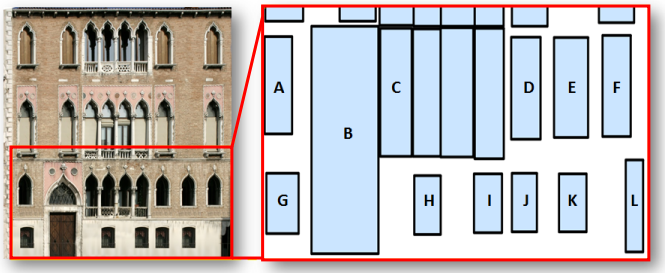


Fig. 1 Complexity and multiformity of constraints in a facade layout. Elements B and C are partially aligned (top aligned, but not bottom aligned). The large element B is aligned with different objects at the top (C ... F) and bottom (L). Elements are missing from a regular pattern consisting of A, D, E, and F. The spacing between same-sized elements H, I, J, K is irregular.

work well together. Fourth, we regularize the layout by transforming the contents of the layout such that the change in both element locations and sizes is minimal while the selected constraints are also respected. Our formulation for layout regularization minimizes energy expenditure from quadratic programming.

In our results, we show that our algorithm performs better than [5] and also better than the independently developed algorithm in [6]. Further, our framework considers more types of constraints than have been considered in previous work. The constraints we consider include the size, spacing, and alignment of layout elements.

We make the following contributions:

- A formulation of the layout regularization problem that performs better than previous work, as evaluated on a test dataset consisting of layouts from a variety of applications.
- An extension of previous work with a larger variety of constraints that can be detected and considered in the layout optimization.

2 Related Work

The layout problem can be roughly classified into two major categories, seamless layouts without gaps and layouts with gaps between elements. Our work focuses on the latter category of layout problem. We review related work in image structure analysis, geometry structure analysis, and layout enhancement.

2.1 Image structure analysis

There is a large literature that addresses different aspects of image structure analysis. A common interest in computer graphics and computer vision is facade layout analysis in urban modeling [7]. The image labeling problem has been addressed by considering both visual evidence and architectural principles [8]. Based on a perceptual grouping approach, translational symmetry is exploited for single view image recognition [9]. A similar approach that uses repeated information for facade image reconstruction is proposed by Wu et al. [10]. To understand the structure of a facade, a set of facade images are first recursively split and labeled for training, and then the features are extracted from the segmented facades and used to guide the labeling. Riemenschneider et al. [11] combine both low-level and mid-level classifiers for irregular facade parsing. Yang et al. [12] use a binary split grammar to parse facade images. Teboul et al. [13] parse facade layouts by using reinforcement learning. Wu et al. [1] extract grammars from labeled facade layouts and generate large scale variations by editing the grammars. Shen et al. [14] adaptively partition urban facades into hierarchical structures based on concatenated and interlaced grids. Musialski et al. [15] developed an interactive tool for facade image segmentation that requires significant amount of user interaction. While most of these analyses of facade layouts use a hierarchical representation, Zhang et al. [16] propose modeling layouts using layered grids with irregular spacing. In our work, we also use grids with irregular spacing, but we can avoid the complexity of the layered structure.

2.2 Geometry structure analysis

Quite a few papers focus on discovering regular patterns for geometry structure analysis in the 3D space. Mitra et al. [17] propose a pair-matching based approach to detect partial and approximate symmetry in 3D shapes. Pauly et al. [3] further introduce a framework for detecting translational, scaling and rotational patterns in 3D shapes. Tevs et al. [18] build a connection among similar shapes via geometric symmetries and regularities. These approaches have inspired many applications in shape analysis, reconstruction, and synthesis. For example, Li et al. [19] propose reconstructing 3D shapes from noisy and incomplete point cloud data that simultaneously detects surface primitives and their mutual relationships. This approach involves both local and global shape analysis. A recent survey paper [20] presents more related

2.3 Layout enhancement

The layout enhancement (regularization and beautification) has been studied in different areas, e.g., object alignment [21], handwriting and drawing beautification [22], [23], [24], sketch and drawing beautification [5], [6], [25], [26], and 3D shape symmetrization [27]. Nan et al. [28] exploit and model conjoining Gestalt rules for grouping and summarization of facade elements. AlHalawani et al. [29] analysze and edit the facade images with (semi-)regular grid structures. Huang et al. [30] combine patchbased image completion and translational symmetry detection to fill in the missing part of an incomplete planar structure. More recently, Xu et al. [31] propose a command-based arrangement tool for 2D layouts.

Pavlidis and Van Wyk [5] beautify drawings using a clustering method, while Xu et al. [6] interactively enhance the global beautification with user guidance. We compare our approach to these two methods in Section 6.

In this work, we are interested in processing digitized two-dimensional (2D) images and drawings. By abstracting each layout as a set of rectangles, our goal is to regularize the layout such that the regularities of the elements in the layout are enforced.

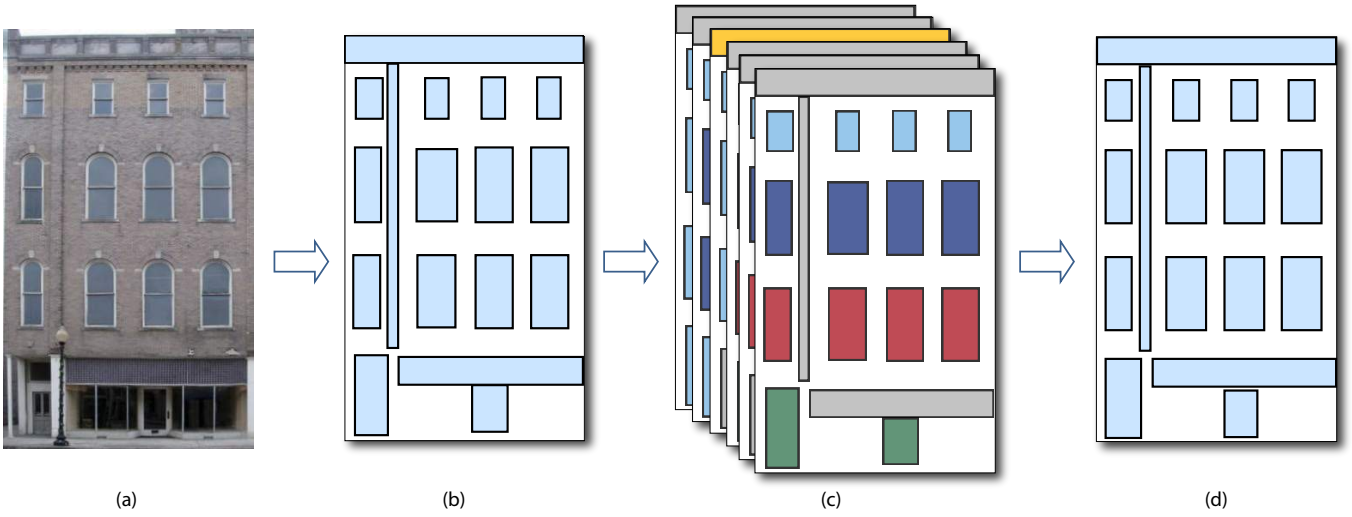


Fig. 2 An overview of our layout regularization approach. Given an input image or drawing (a), we first obtain the initial layout by manually marking and labeling the elements in a pre-processing step (b). Then, appropriate constraints are automatically selected (c) and are used to generate the regularized layout (d).

3 Overview

Given an image or drawing $\mathbf{I}$, that is characterized by a set of rectangles, the layout $L = \{e_1, ...e_n\}$ of $\mathbf{I}$ can be simply described as the locations and sizes of the elements in $\mathbf{I}$. Here, an element, e_i, is defined by a label, l_i, and its bounding box, $b_i = \{x_i, y_i, w_i, h_i\}$ depicting its bottom-left corner (x_i, y_i) and the size (w_i, h_i) (see Fig. 4). We seek to regularize the layout of the elements such that the regularities of these elements are enforced.

Our proposed solution to the layout regularization problem uses both discrete and continuous optimization. Fig. 2 shows an overview of our layout regularization method. Our method consists of the following three steps.

3.1 Preprocessing

To digitize the layout of a given image, the user manually marks and labels the elements in the input image. The output of the preprocessing step is the initial layout that will be regularized in the next steps. Alternatively, the input can be user-generated drawings or slide layouts.

3.2 Constraint selection

We first detect a larger set of candidate constraints from the initial layout using a simple thresholding-based method. Then, we score each constraint using an energy function. Finally, we select a set of constraints from the candidates using global optimization (linear integer programming). Details on constraint selection are presented in Section 4.

3.3 Layout regularization

To regularize the input layout, we transform the contents of the layout such that changes in both the element locations and sizes are minimal while selected constraints are respected. We use quadratic programming to minimize energy use in layout regularization (Section 5).

4 Constraints Selection

Given user-marked elements in a layout, our layout regularization method tries to detect and enforce three types of constraints: alignment, same-size, and same-spacing constraints. This problem is challenging in the following ways. First, we have to detect reasonable constraints connecting elements in the layout. Second, there may exist potential conflicts among these constraints. To address these problems, we introduce an optimization-based constraint selection algorithm. The selected constraints are then used in a quadratic programming formulation to regularize the layout (see Section 5).

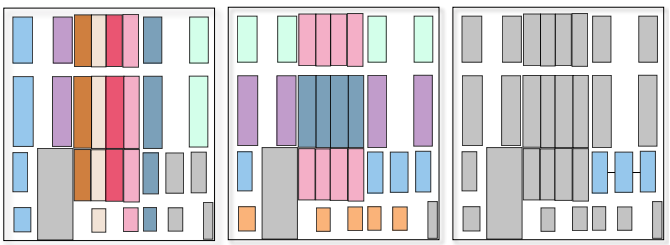


Fig. 3 A subset of constraints in the example layout in Fig. 1. Colors indicate different constraint groups in this figure.

4.1 Constraint Definitions

We consider the following relationships between elements as potential regularity constraints: alignment constraints, same-size constraints, and same-spacing constraints (see Fig. 3 and Fig. 4).

4.1.1 *Alignment constraints*

Two elements, e_i and e_j can have one or multiple of the following alignment constraints: top alignment, middle-Y alignment, bottom alignment, left alignment, middle-X

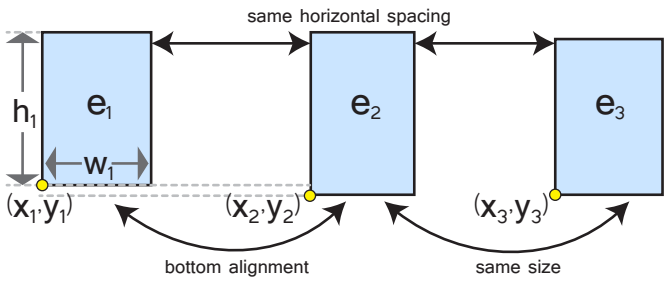


Fig. 4 Illustration of three types of constraints. The bottom alignment of elements e_1 and e_2 can be formulated as $y_1 - y_2 = 0$. For the same-size constraint of element pair (e_2, e_3), we have $w_2 - w_3 = 0$ and $h_2 - h_3 = 0$. The horizontal same-spacing constraint on element pairs $\{e_1, e_2\}$ and $\{e_2, e_3\}$ turns out to be $x_2 - (x_1 + w_1) - (x_3 - (x_2 + w_2)) = 0$.

alignment, and right alignment. For example, a bottom alignment between e_i and e_j can be formulated as

$$y_i - y_j = 0. \tag{1}$$

Other alignment relations are defined in a similar way.

4.1.2 Same-size constraints

Two elements, e_i and e_j, may be linked by a same-width constraint or a same-height constraint or both. Elements with the same label are always considered to hold same-size constraints. Same-size constraints can be formulated as:

$$\begin{aligned} w_i - w_j &= 0, \\ h_i - h_j &= 0. \end{aligned} \tag{2}$$

4.1.3 Same-spacing constraints

Same-spacing constraints are defined on two-element pairs and can be either in the horizontal or vertical direction. Currently, we only consider same-spacing constraints between elements with the same labels. For example, assume the element pairs (e_i, e_j) and (e_m, e_n) should have the same spacing in the horizontal direction. The equations for same-spacing constraints depend on the relative position of the elements. For the given example, assuming $x_i < x_j$ and $x_m < x_n$ would lead to

$$x_i + w_i - x_j - (x_m + w_m - x_n) = 0. \tag{3}$$

4.2 Candidate Group Generation

The candidate group generation step computes a set of candidate groups $\{g_i\}$, where each group, g_i, is a set of elements that share an alignment, same-size, or same-spacing constraint. In this step, we use a threshold, t_a, to limit the candidate groups to a reasonably small set. Note that the threshold, t_a, is a global control mechanism for the number of candidate groups being generated. This threshold is set high enough so that all reasonable candidates are generated. Note that t_a is only used for generating the candidate constraints, while the actual constraints are selected using the linear integer programming formulation described later. We describe the candidate group generation for each constraint type in the following.

4.2.1 Alignment constraints

We use top-alignment as an example. We sort all the elements in the input layout according to the y value of their top edge. Let $\{p_1, ...p_n\}$ denote the top positions of the sorted elements. We generate a set of potential groups such that the difference in the top positions of every pair of the elements in each group is less than the threshold, t_a.

4.2.2 Same-Size constraints

For same-size constraints, we first group all elements according to their label because we assume that elements with the same label have the same size. Then, we compute the average element size for each label and use this element size to define a distance between labels using the l_1 norm. For each label we find the k-nearest neighboring labels, where k iterates from 1 to the number of labels. This yields an initial set of candidate groups. Then, we filter out candidate groups in which there exist two elements with a size difference larger than the threshold, t_a.

4.2.3 Same-Spacing constraints

We take horizontal same-spacing as an example. We sort all the element pairs in the input layout according to their horizontal intervals. Then, the same-spacing constrained groups are generated by combining element pairs so that each group satisfies the following two conditions: 1) The difference in the interval of every element pair is less than the threshold, t_a; 2) The elements overlap in the vertical direction.

4.3 Energy Functions

We now describe how to assign energy values to candidate groups such that groups with lower energies are more likely to be selected. We first describe a set of auxiliary heuristic functions that will then be combined to obtain various energy functions. In the optimization, we will use a linear combination of the described energy functions as the objective function. A constraint group, g_i, is composed of a set of elements, $\{e_1, ..., e_n\}$ (two-element pairs for same-spacing). We define the following functions on g_i:

4.3.1 Standard deviation

The function $stdvar(g_i)$ measures the standard deviation of positions (for alignment constraints), sizes (for same-size constraints), or spacings (for same-spacing constraints). For example, if g_i is a group of top-aligned elements, $stdvar(g_i)$ is the standard deviation of the top positions of all elements in group g_i.

4.3.2 Maximal element distance

The function $maxDist(g_i)$ computes the maximal distance between positions, sizes, or spacings. For example, for a group of top-aligned elements, $maxDist(g_i)$ is defined as the difference between the maximal and the minimal top position in the group.

4.3.3 Group scale

The function $scale(g_i)$ is an intuitive measure for the scale of group g_i in the relevant direction (x or y). This function is evaluated differently for alignment, same-size, and same-spacing constraints. For example, for a group, g_i, with horizontal alignment, $scale(g_i)$ is equal to the minimal height of elements in group g_i. For same-size constraints, $scale(g_i)$ is the maximum of the minimal width and minimal height of the elements in group g_i. For same-spacing constraints, we define $scale(g_i)$ as the minimum spacing between element pairs in g_i.

To measure the quality of a constraint group, we consider the following energy terms.

4.3.4 Intra-group distance

In our analysis, a good group should have a small variance and a small maximal element distance. Further, these values should be normalized by scale:

$$E_d(g_i) = \frac{max(0, stdvar(g_i) + maxDist(g_i) - \epsilon)}{scale(g_i)}, \quad (4)$$

where ϵ is the maximal allowed tolerance value so that distances smaller than ϵ will be ignored. We set ϵ to 3 pixels based on our experiments.

4.3.5 Aspect ratio variance

For same-size constraints, the aspect ratio plays an important role. Thus, we use an energy term, $E_a(g_i)$, that captures the standard deviation of the aspect ratio of all elements in group g_i. Here, the aspect ratio of an element is defined as $\frac{w}{h}$, where w and h are the width and height of the element.

4.4 Constraint Selection

We employ linear integer programming to select a set of constraint groups among the candidate groups. There are multiple goals: First, the energy values of the selected groups should be low. Second, the complexity of the overall model measured by the number of constraint groups used should also be low. This motivates the use of an additional sparsity term. In our formulation, each constraint type uses a different energy function.

Given an input layout, L, consisting of n elements and the candidate constraint groups, $G = \{g_1, ..., g_N\}$, generated from L, our task is to choose a subset of these candidate groups as constraints for the following layout regularization step. Let $C = C_a \bigcup C_{ss} \bigcup C_{sp}$ denote all the constraint types, where C_a, C_{ss}, and C_{sp} are alignment, same-size, and same-spacing types, respectively. $\mathbf{Z} = \{z_1, ...z_N\}$ denotes the binary label for each candidate group (1 for *chosen* and 0 for *not chosen*). We split $\mathbf{Z}$ into three subvectors, $\mathbf{Z_a}$, $\mathbf{Z_{ss}}$, and $\mathbf{Z_{sp}}$, representing the labels for each type of constraint groups. Then, the energy functions for these types of constraint group are defined as follows:

4.4.1 Alignment constraints

$$E(\mathbf{Z_a}) = \sum_{c_j \in C_a} \sum_{g_i \in G} E_d(g_i) \cdot z_i \cdot \delta(g_i, c_j) + w_a \cdot \|\mathbf{Z_a}\|_0, \quad (5)$$

where $\|\cdot\|_0$ denotes the ℓ^0-norm, which counts the number of nonzero entries in a vector. We add this term to encourage fewer and larger groups (i.e., groups that have more elements). Because $z_i \in \{0, 1\}$ in our problem, $\|\cdot\|_0$ can be simplified to the sum of all the entries in the vector. $\delta(g_i, c_j)$ is an indicator function that has value 1 if g_i is a candidate group of constraint type c_j; otherwise it is zero. w_a is a weight that balances the two terms.

4.4.2 Same-Size constraints.

The energy function for same-size constraints is similar to that for alignment constraints. To account for aspect ratio of an element in the layout, we also involve the aspect ratio variance E_a into the formulation:

$$E(\mathbf{Z_{ss}}) = \sum_{c_j \in C_{ss}} \sum_{g_i \in G} (E_d(g_i) + w_a \cdot E_a(g_i)) \cdot z_i \cdot \delta(g_i, c_j) + w_{ss} \cdot \|\mathbf{Z_{ss}}\|_0, \quad (6)$$

4.4.3 Same-Spacing constraints.

For same-spacing constraints, the energy function is similar to that of alignment constraints:

$$E(\mathbf{Z_{sp}}) = \sum_{c_j \in C_{sp}} \sum_{g_i \in G} E_d(g_i) \cdot z_i \cdot \delta(g_i, c_j) + w_{sp} \cdot \|\mathbf{Z_{sp}}\|_0, \quad (7)$$

Afterwards, proper constraint groups are selected by minimizing the following constrained objective function:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & E(\mathbf{Z_a}) + E(\mathbf{Z_{ss}}) + E(\mathbf{Z_{sp}}) \\ \text{subject to} \quad & \sum_{i=1}^{N} \|g_i\| \cdot z_i \cdot \delta(g_i, c_j) = n, \quad c_j \in C \\ & z_i + z_j \leq 1, \quad \forall g_i \bigcap g_j \neq \emptyset, \quad 1 \leq i, j \leq N \\ & z_i \in \{0, 1\}, \quad 1 \leq i \leq N, \end{aligned} \quad (8)$$

where the constraints $\sum_{i=1}^{N} \|g_i\| \cdot z_i \cdot \delta(g_i, c_j) = n$ ensure that every element in the layout is assigned to a constraint group of type c_j. The second group of constraints, $z_i + z_j \leq 1$, ensure that groups do not have overlapping elements if g_i and g_j are of the same constraint type.

The optimization problem above is a linear integer program that can be efficiently solved using various open source solvers, e.g., [32], [33], [34]. The solution is a set of constraint groups. Each group gives rise to a set of linear equations that serve as constraints during the layout regularization step. For example, for an alignment group $g_i = \{e_{i1}, ..., e_{iN}\}$, we combine adjacent elements to form the constraint pairs, namely, $(e_{i1}, e_{i2}), ..., (e_{i(n-1)}, e_{iN})$. Then, we generate one linear equation per constraint pair.

5 Layout Regularization

With the optimal constraints detected and filtered from the constraint selection step, our final goal is to regularize the layout under these constraints. Our regularization process has a similar format as the methods presented in [35] and [36]. These works emphasize the facade structure using a hierarchical layout, while ours deals with a layout of rectangles. We address this regularization problem by transforming the contents of the layout from the input layout, L, to a regularized layout, L^*, such that the change to element locations C_L and element sizes C_S is minimal

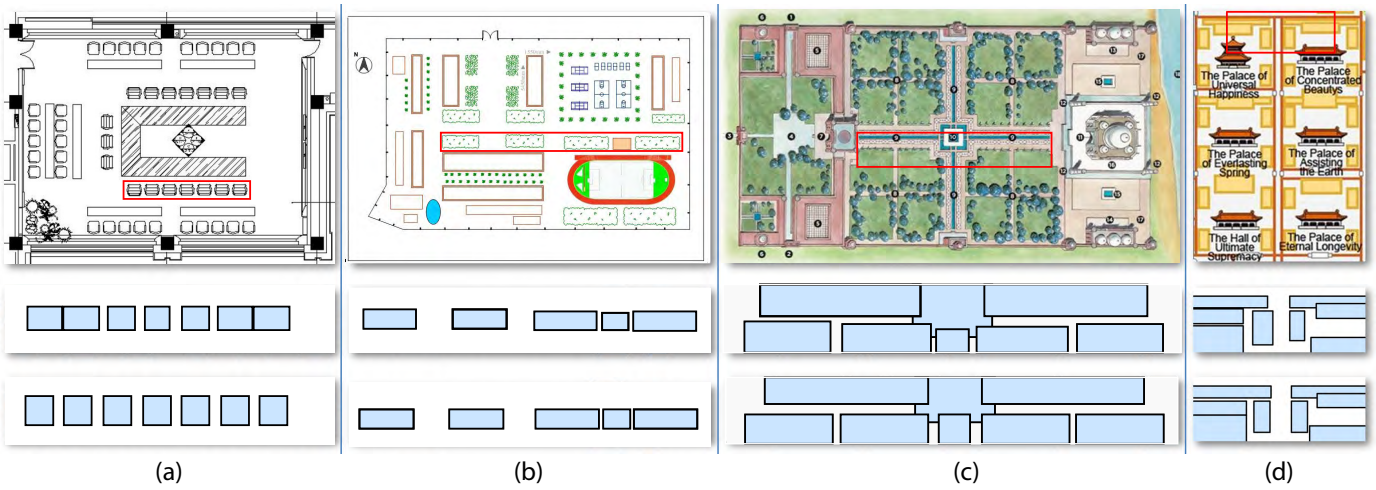


Fig. 5 Four different layouts are regularized using our method. Each column (from top to bottom) shows the input floor plan, zoom in views of the marked region in the initial layout, and our regularized layout.

while respecting the constraints. The star ($*$ in L^*) indicates the regularized layout.

To facilitate user preferences, we use a weight, ω (we set it to 2.5 for our preference for position changes), to balance between the two terms above. Then, the layout regularization is formulated as an energy minimization problem as below:

$$L^* = arg\min\ (C_L + \omega \cdot C_S), \tag{9}$$

where

$$C_L = \sum_{i=1}^{n}(x_i^* + \frac{w_i^*}{2} - x_i - \frac{w_i}{2})^2 + (y_i^* + \frac{h_i^*}{2} - y_i - \frac{h_i}{2})^2$$

$$C_S = \sum_{i=1}^{n}(w_i^* - w_i)^2 + (h_i^* - h_i)^2.$$

In addition to the aforementioned constraints selected in Section 4, we add additional constraints to Equation 9 to ensure the validity of the optimized layout. In our formulation, we include lower bound constraints and upper bound constraints for the variables and sequential constraints for the relative positions of the elements. These constraints are as follows:

- **Lower and upper bound constraints**. These constraints restrict changes in elements in reasonable ranges. Let (w_b, h_b) denote the size of the bounding box of the layout. We add additional positional constraints, $0 \le x_i^* \le w_b$ and $0 \le y_i^* \le h_b$.
  Further, to prevent the size of the elements from being changed too much, we also add upper bound constraints on their sizes. Let's take the width bound as an example, it is defined proportionally to the widths of all elements that have the same label, ℓ. In our implementation, the maximal allowed width change for an element, e_i, is defined as $max(0.5 \cdot \Delta w_\ell, 0.15 \cdot w_i)$, where Δw_ℓ is the maximal difference in width for elements that have the same label, and w_i is the width of e_i. The size constraints on element height are defined similarly.
- **Sequential constraints**. These constraints specify the relative positions of pairs of elements. With these constraints, we expect that the original layout of the elements will not be greatly altered by the regularization. Our experiments show that this type of constraint is crucial to layout regularization. Given two X-ordered (ascending order) elements, e_i and e_j, the constraints are $x_i^* + w_i^* - x_j^* \le 0$ if $x_i + w_i - x_j \le 0$. The same goes for the vertical direction.

By solving the quadratic programming problem defined in Equation 9, we obtain the regularized layout. In our implementation, we add the constraints sequentially to avoid potential conflicts. If any conflict is detected during the optimization, we simply remove the current constraint. However, the sequence of constraints will affect the results. To incorporate our preferences for different constraints, we sort all constraints according to their energy function, $stdvar(g_i)$ (see Eq. 4), and then we add them to the constraint set according to this order.

6 Results and Discussion

6.1 Test Database

Our experiments are conducted on a database of 32 digitized layouts from various applications. Our data set contains examples covering facades, slide designs, web designs, indoor scenes, and other graphical layouts. In Figures 5, 6, and 7, we show a set of different layouts regularized using our method. In the supplemental materials, we provide more results showing detected relations and regularized layouts. From these applications, we can see that our method enforces the regularity constraints, while preserving high-level relations, such as symmetries and repetitive patterns.

6.2 Evaluation Metrics

To evaluate the effectiveness of our framework, we design an interactive program to specify the ground truth relationships for each layout. We use the marked relations to compute precision (P), recall (R), and F-measure (F), defined as follows:

Fig. 6 Layout regularization for a set of urban facade images. The top row shows the input facade images. The middle and the bottom rows show the zoom in views of the highlighted regions and the regularized results with abstract boxes.

$$
\begin{aligned}
P &= \frac{\sum_{i \in G_g} \sum_{j \in G_d} num(g_i \bigcap g_j)}{\sum_{j \in G_d} num(g_j)}, \\
R &= \frac{\sum_{i \in G_g} \sum_{j \in G_d} num(g_i \bigcap g_j)}{\sum_{i \in G_g} num(g_i)}, \\
F &= \frac{2 \cdot P \cdot R}{P + R},
\end{aligned}
\tag{10}
$$

where G_g is the set of constraint groups in the ground truth, G_d is the set of constraint groups in the detected result, and $num(\cdot)$ is the number of constraints in a constraint group. The term $g_i \bigcap g_j$ denotes the intersection of two constraint groups. It is empty if g_i and g_j are different types of constraint. For alignment and same-size constraints, an $n-$element constraint group will contribute $n-1$ constraint pairs, while an $n-$pair spacing group will contribute $n-1$ constraint pairs (see Section. 4.1). Thus, we define the number of constraints of a constraint group as the number of constraint pairs it yields. For example, consider that we have the top alignment of elements $G = \{e_1, e_2, e_3, e_4, e_5\}$ as ground truth, but the algorithm detects only elements $D = \{e_1, e_2, e_3, e_5\}$ as top aligned. Then, we have $num(D) = 4-1$, $num(G) = 5-1$, and $num(G \bigcap D) = 4-1$.

6.3 Comparison

We made comparisons with the methods of Xu et al. [6] and Pavlidis et al. [5]. Pavlidis et al. [5] propose to use a clustering method to detect the constraints. Xu et al. [6] employ the RANSAC method to get alignment constraints, and a clustering method for same-spacing detection. We first conduct a test of the alignment constraints. Fig. 8 shows the precision, recall, and F-measure for every layout in our test database. As we can see, our algorithm has similar precision to previous work, but it has higher recall for most layouts. This leads to the highest F-measure results on 93.8% of layouts in the database. The comparison is also summarized in Table 1. In the next test, we compare the same-spacing constraints. In Table 1, we present a comparison of the average values for precision, recall and F-measure. From this comparison, we see that the same-spacing constraint is more difficult to detect than is the alignment constraint. Additionally, it is very difficult to define a ground truth for this constraint. From the above comparison, we can see that the precision of the same-spacing detection benefits from the label information. However, our method works better than others even without the label information. In Fig. 9, we also show an illustrative example of a case where our method is more successful than Xu et al.'s [6]. We can see that Xu et al. can not handle layouts in which elements overlap with each other. Their method cannot align the elements properly. In addition, we compare the performance of these three methods by measuring the average computation time for the examples in our dataset. The method in [5] detected the constraints in $0.001s$ because of the simplicity of the method. Xu et al.'s method [6] needs $0.898s$, while ours needs about $0.914s$.

6.4 Running Time

We implement the proposed method in C++. All the experiments are performed on a PC with two 2.70GHz Intel Xeon E5-2680 processors. We find that the running time depends on the number of elements and the number of relations in the input layout. On average, the constraint selection step takes $0.70s$, and the regularization step takes $3.59s$.

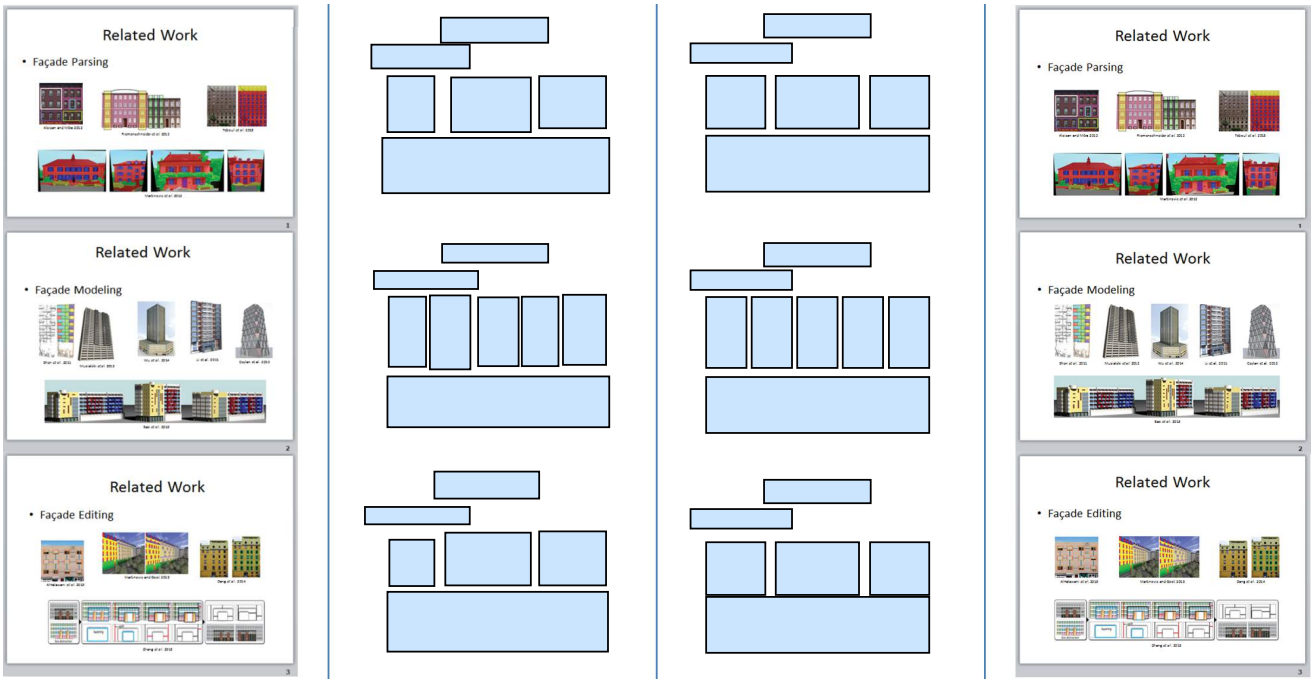


Fig. 7 Slide design beautified using our approach. From left to right: (a) the initial design, (b) the bounding boxes of the elements in the design as input layout, (c) regularized layout, and (d) the final design.

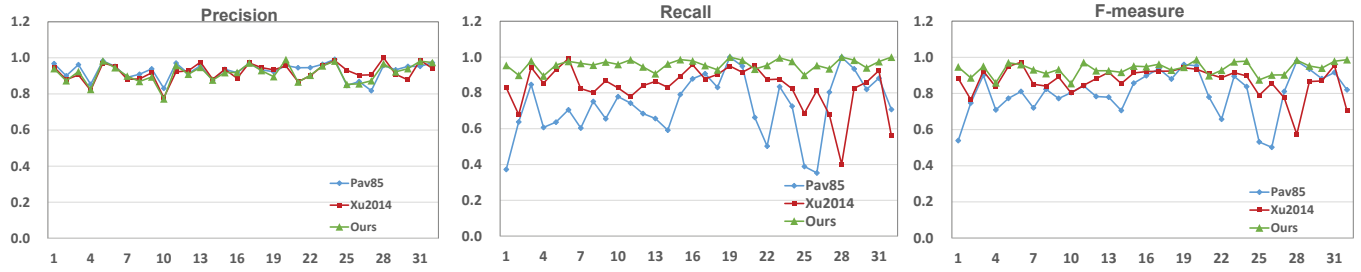


Fig. 8 The comparison of precision (left), recall (middle), and F-measure (right) of our method with Pav85 [5] and Xu2014 [6] on the alignment constraints.

TABLE 1 Comparisons with [5] and [6] on alignment and same-spacing constraints. Our method is evaluated with and without labels. We show the average precision (P), recall (R), and F-measure (F) for all layouts in the test dataset.

Method	Alignment			Same-spacing		
	P	R	F	P	R	F
Pav85 [5]	**0.927**	0.726	0.804	0.782	0.366	0.453
Xu2014 [6]	0.920	0.832	0.868	0.732	0.498	0.540
Ours (no label)	0.911	0.959	0.936	0.750	**0.706**	**0.710**
Ours	0.911	**0.959**	**0.936**	**0.916**	0.613	0.696

The maximum times for these steps were $3.71s$ and $15.78s$, respectively.

6.5 Robustness and Scalability

We evaluate the robustness and scalability of our algorithm on synthesized examples. We first generate a regular grid of elements of two different sizes with 8 columns and 5 rows. We then perturb the corners of the elements with an increasing amount of Gaussian noise (measured relative to the element sizes). The performance of our method is demonstrated in Table 2. We can see that our method works well if the noise is less than 10% of the element size. To evaluate the scalability, we use Gaussian noise with a variance of 0.02 and measure the running time for grids with a different number of elements. In Table 3, we present the results of this test. We can see that the accuracy decreases with larger grids. The reason for these decrease is mainly that some of the same-spacing constraints are not detected due to outliers.

6.6 Parameters.

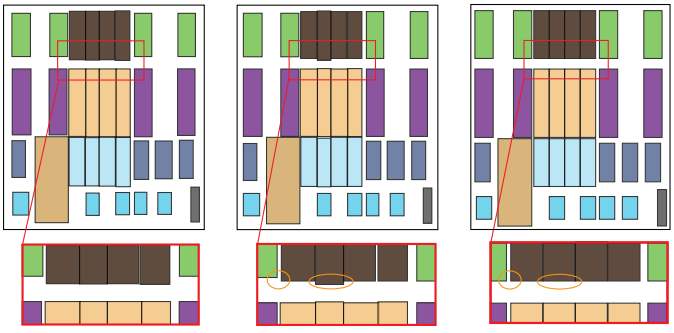
Our method includes multiple parameters. One parameter is the threshold, t_a, that is used to generate candidate groups. To verify the influence of this parameter on the results, we evaluate our method with different values of the threshold, t_a, on the alignment constraints (see Fig. 10). Our method

TABLE 2 Performance of our algorithm on a data set with increasing amount of Gaussian noise relative to the element size. #C is the number of detected constraints.

Level of noise	#C	P	R	F
0.00	321	1.000	1.000	1.000
0.02	321	1.000	1.000	1.000
0.04	319	1.000	0.993	0.996
0.06	297	1.000	0.915	0.956
0.08	290	1.000	0.894	0.944
0.10	263	1.000	0.795	0.886

TABLE 3 Performance of our algorithm on a data set with an increasing of number of rows and columns. Note that all the grid elements are perturbed by 2% Gaussian noise (relative to the element sizes).

Grid size	#C	P	R	F	Time(s)
5×8	321	1	1	1	2.245
10×8	678	0.987	0.983	0.985	6.428
5×16	676	0.975	0.986	0.981	6.996
10×16	1420	0.964	0.971	0.967	25.789
20×16	2940	0.947	0.964	0.955	121.822



(a) Input layout (b) Xu et al. (c) Ours

Fig. 9 A comparison of Xu et al. [6] (b) and our method (c). The yellow circles indicate the differences.

can generate high-quality results after a value of 0.2 times the average element size, which makes our method reliable even without user intervention. Another important parameter is the weight of the sparsity term. Here, we evaluate the performance with respect to the sparsity term w_a as shown in Fig. 11. The sparsity term plays an important role in selecting the trade off between precision and recall.

6.7 Applications.

Our method is designed for general 2D layouts. One application is the regularization of digitized layouts, e.g., the facade layouts shown in Fig. 6. Another application is the beautification of user-drawn layouts, e.g., slide design (see Fig. 7), poster design, and other graphical designs (see Fig. 5).

6.8 Extensions.

Our current implementation is developed for axis-aligned layouts, but we can extend our framework to consider more types of constraints and elements enclosed in oriented

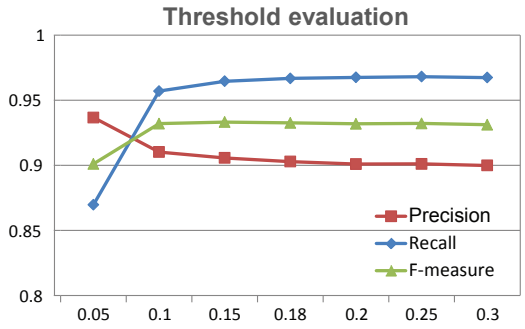


Fig. 10 The robustness of our method with respect to the threshold, t_a. We show the change in average precision, recall, and F-measure for the alignment threshold uniformly sampled in the range [0.05, 0.3].

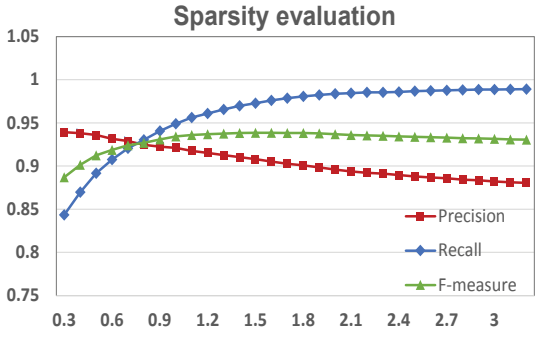
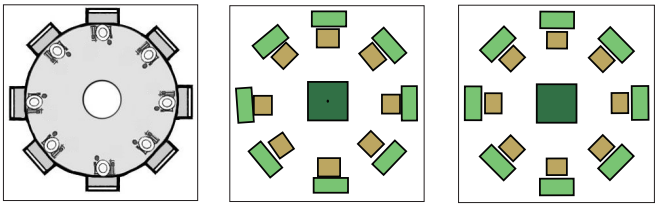


Fig. 11 The robustness of our method with respect to the sparsity term, w_a.

bounding boxes. In Fig. 12, the elements are distributed on circles. For this example, we introduce two new types of constraints that consider spacing and alignment in radial layouts. Our algorithm can be directly used for these constraints with a polar coordinate system.

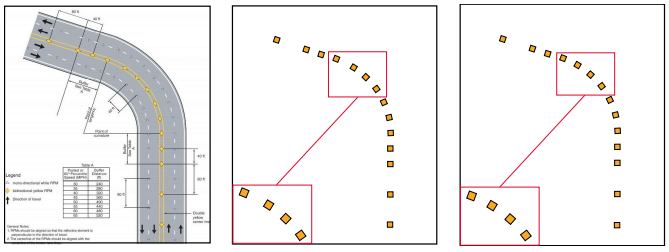


(a) Initial design (b) Initial layout (c) Regularized layout

Fig. 12 An extension of our algorithm. In this example, elements (i.e., chairs or the tableware) are expected to be placed along concentric circles with same included angles. The black dot in (b) indicates the center of circles. (c) shows the result of our algorithm applied to this case by using a simple coordinate system conversion (from the Cartesian coordinate system to the polar coordinate system).

Another type of useful constraint is the same-arc-length distance constraint. The same-arc-length constraint enforces a constant arc length along a curve between two adjacent points that are sampled on this curve. In Fig. 13(a), we show a set of markers (the yellow squares) that are placed along

the centerline of the road (in yellow). We can see that some adjacent markers exhibit the same-arc-length constraint. Directly fulfilling this constraint is difficult, considering that we do not know the curve function. We construct a map from a parameter vector to the points by a B-spline interpolation with chord length parameterization. Thus, every parameter corresponds to a point, and the interval between two parameters is equivalent to the chord length of two adjacent points. Then, we achieve the same arc-length by accomplishing the same spacing constraint on the parametric vectors. In Fig. 13(c), we show the result of this regularization. Another example is given in Fig. 6 of the supplemental materials.



(a) Initial design (b) Initial layout (c) Regularized layout

Fig. 13 The same-arc-length distance constraint on points (yellow squares). Our method successfully detects two kinds of arc-length in this example and regularizes the curve points. Our framework can be applied to such input by constructing a parameterization of the points.

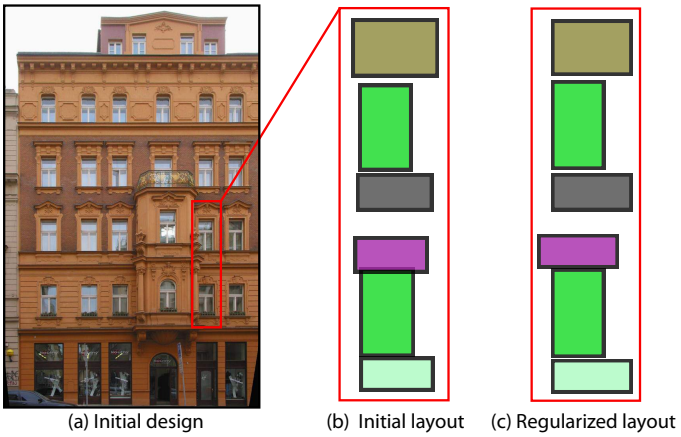
Our method does not fully explore the hierarchical structuring of constraints. However, we are able to consider a case in which a given hierarchy defines the grouping information of elements (see Fig. 14). The regularization is achieved by applying our method from bottom to top.

6.9 Limitations and future work.

Although our algorithm works well on most cases, we notice that in some cases the result could be further improved with the availability of semantic information. For example, if there is an ornament on top of a window we can assume that there is a high probability that the two shapes are center aligned (see Fig. 15). Not using domain-specific semantic priors is one limitation of our algorithm. Another limitation is that possible constraints need to be known in advance. when these is a large number of complex patterns, e.g., a set of elements aligned along a spiral with regularly decreasing spacing, it is unclear how our framework would perform if we would extend it using a large number of different complex constraint types. We consider this a very interesting avenue of future work. Further, we also plan to involve users in the layout optimization stage to provide more control over the regularization process.

7 Conclusions

We have presented an optimization-based approach for regularizing general layouts. Our method takes as input a general layout represented as a set of labeled rectangles,



(a) Initial design (b) Initial layout (c) Regularized layout

Fig. 15 A failure case of our algorithm. In this example, the user marks a wrong left edge of the ornaments below the windows in the highlighted region due to occlusions caused by perspective projection. Semantic prior information (e.g., an ornament and window are more likely to be center aligned) is necessary to correct this error.

and detects regularity constraints based on a linear integer programming formulation. The layout is regularized by minimizing the deformation of the initial layout while respecting the detected constraints. We have evaluated our method using various input layouts. Experimental results show that our method enforces the regularities in the layout and that it is superior to alternative approaches in the literature. We have also shown the usefulness of our method in various applications.

Acknowledgments

We would like to thank the reviewers for their helpful comments and the authors of [6] for making their software publicly available and for their help on the comparison. This work was supported by the KAUST Visual Computing Center, the China National 863 Program (No. 2015AA016402), the National Natural Science Foundation of China (No. 61372168, 61331018, 61372190, and 61272327), and the U.S. National Science Foundation.

References

[1] F. Wu, D.-M. Yan, W. Dong, X. Zhang, and P. Wonka, "Inverse procedural modeling of facade layouts," *ACM TOG (SIGGRAPH)*, vol. 33, no. 4, pp. 121:1–121:10, Jul. 2014.

[2] C. Wu, J.-M. Frahm, and M. Pollefeys, "Detecting large repetitive structures with salient boundaries," in *ECCV*. Springer, 2010, pp. 142–155.

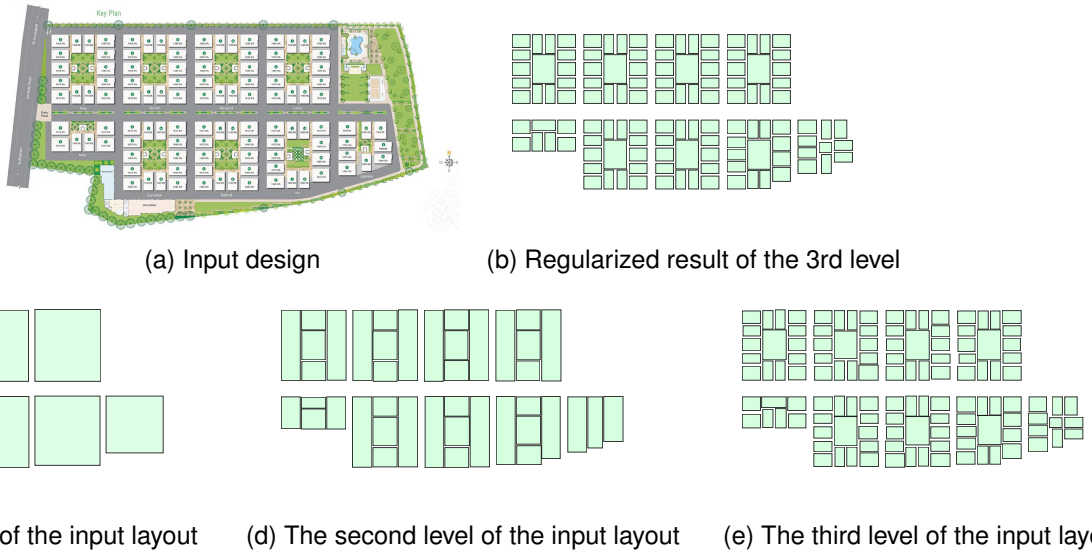
[3] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. Guibas, "Discovering structural regularity in 3D geometry," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 43:1–43:11, 2008.

[4] N. J. Mitra, A. Bronstein, and M. Bronstein, "Intrinsic regularity detection in 3d geometry," in *ECCV*. Springer, 2010, pp. 398–410.

[5] T. Pavlidis and C. J. Van Wyk, "An automatic beautifier for drawings and illustrations," *Computer Graphics (Proc. SIGGRAPH)*, vol. 19, no. 3, pp. 225–234, Jul. 1985.

[6] P. Xu, H. Fu, T. Igarashi, and C.-L. Tai, "Global beautification of layouts with interactive ambiguity resolution," in *UIST '14*, 2014.

[7] P. Musialski, P. Wonka, D. G. Aliaga, M. Wimmer, L. van Gool, and W. Purgathofer, "A survey of urban reconstruction," *Computer Graphics Forum*, vol. 32, no. 6, pp. 146–177, 2013.



(a) Input design (b) Regularized result of the 3rd level

(c) The first level of the input layout (d) The second level of the input layout (e) The third level of the input layout

Fig. 14 The regularization of a hierarchical layout. The second row shows the hierarchy from top to down, which is marked by the user. We use only the marked hierarchy to define the group information of lower-level layouts.

[8] D. Dai, M. Prasad, G. Schmitt, and L. Van Gool, "Learning domain knowledge for facade labelling," in *ECCV*, 2012, pp. 710–723.

[9] M. Park, K. Brocklehurst, R. T. Collins, and Y. Liu, "Translation-symmetry-based perceptual grouping with applications to urban scenes," in *ACCV*, 2011, pp. 329–342.

[10] C. Wu, J.-M. Frahm, and M. Pollefeys, "Repetition-based dense single-view reconstruction," in *CVPR*, 2011, pp. 3113–3120.

[11] M. Donoser, "Irregular lattices for complex shape grammar facade parsing," in *CVPR*, ser. CVPR '12, 2012, pp. 1640–1647.

[12] C. Yang, T. Han, L. Quan, and C.-L. Tai, "Parsing facade with rank-one approximation," in *CVPR*, 2012, pp. 1720–1727.

[13] O. Teboul, I. Kokkinos, L. Simon, P. Koutsourakis, and N. Paragios, "Parsing facades with shape grammars and reinforcement learning," *IEEE PAMI*, vol. 35, no. 7, pp. 1744–1756, 2013.

[14] C.-H. Shen, S.-S. Huang, H. Fu, and S.-M. Hu, "Adaptive partitioning of urban facades," *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH ASIA 2011)*, vol. 30, no. 6, pp. 184:1–184:9, 2011.

[15] P. Musialski, M. Wimmer, and P. Wonka, "Interactive coherence-based facade modeling," *Comp. Graph. Forum*, vol. 31, no. 23, pp. 661–670, May 2012.

[16] H. Zhang, K. Xu, W. Jiang, J. Lin, D. Cohen-Or, and B. Chen, "Layered analysis of irregular facades via symmetry maximization," *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2013)*, vol. 32, no. 4, pp. 121:1–121:10, 2013.

[17] N. J. Mitra, L. Guibas, and M. Pauly, "Partial and approximate symmetry detection for 3d geometry," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 25, no. 3, pp. 560–568, 2006.

[18] A. Tevs, Q. Huang, M. Wand, H.-P. Seidel, and L. Guibas, "Relating shapes via geometric symmetries and regularities," *ACM TOG*, vol. 33, no. 4, pp. 119:1–119:12, Jul. 2014.

[19] Y. Li, X. Wu, Y. Chrysanthou, A. Sharf, D. Cohen-Or, and N. J. Mitra, "Globfit: Consistently fitting primitives by discovering global relations," *ACM Transactions on Graphics*, vol. 30, no. 4, pp. 52:1–52:12, 2011.

[20] N. J. Mitra, M. Wand, H. Zhang, D. Cohen-Or, and M. Bokeloh, "Structure-aware shape processing," in *EUROGRAPHICS State-of-the-art Report*, 2013.

[21] P. Baudisch, E. Cutrell, K. Hinckley, and A. Eversole, "Snap-and-go: Helping users align objects without the modality of traditional snapping," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2005, pp. 301–310.

[22] S. Murugappan, S. Sellamani, and K. Ramani, "Towards beautification of freehand sketches using suggestions," in *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, 2009, pp. 69–76.

[23] C. L. Zitnick, "Handwriting beautification using token means," *ACM TOG (SIGGRAPH)*, vol. 32, no. 4, pp. 53:1–53:8, Jul. 2013.

[24] P. O'Donovan, A. Agarwala, and A. Hertzmann, "Learning Layouts for Single-Page Graphic Designs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 8, pp. 1200–1213, 2014.

[25] B. Plimmer and J. Grundy, "Beautifying sketching-based design tool content: Issues and experiences," in *Proceedings of the Sixth Australasian Conference on User Interface - Volume 40*, 2005, pp. 31–38.

[26] B. Paulson and T. Hammond, "Paleosketch: Accurate primitive sketch recognition and beautification," in *Proceedings of the 13th International Conference on Intelligent User Interfaces*, 2008, pp. 1–10.

[27] N. J. Mitra, L. Guibas, and M. Pauly, "Symmetrization," *ACM TOG (SIGGRAPH)*, vol. 26, no. 3, pp. 63:1–63:8, 2007.

[28] L. Nan, A. Sharf, K. Xie, T.-T. Wong, O. Deussen, D. Cohen-Or, and B. Chen, "Conjoining gestalt rules for abstraction of architectural drawings," *ACM TOG (SIGGRAPH Asia)*, vol. 30, no. 6, 2011.

[29] S. AlHalawani, Y.-L. Yang, H. Liu, and N. J. Mitra, "Interactive facades: Analysis and synthesis of semi-regular facades," *Computer Graphics Forum (Eurographics)*, vol. 32, no. 22, pp. 215–224, 2013.

[30] J.-B. Huang, S. B. Kang, N. Ahuja, , and J. Kopf, "Image completion using planar structure guidance," *Proc. ACM SIGGRAPH*, vol. 33, no. 4, p. 129, 2014.

[31] P. Xu, H. Fu, C.-L. Tai, and T. Igarashi, "GACA: Group-aware command-based arrangement of graphic elements," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15, 2015, pp. 2787–2795.

[32] Lpsolve, http://lpsolve.sourceforge.net/.

[33] CBC, https://projects.coin-or.org/Cbc.

[34] GLPK, http://www.gnu.org/software/glpk/.

[35] M. Dang, D. Ceylan, B. Neubert, and M. Pauly, "Safe: Structure-aware facade editing," *Computer Graphics Forum (Eurographics)*, vol. 33, no. 2, pp. 83–93, 2014.

[36] F. Bao, M. Schwarz, and P. Wonka, "Procedural facade variations from a single layout," *ACM Trans. Graph.*, vol. 32, no. 1, pp. 8:1–8:13, 2013.