Hierarchical Layout Blending with Recursive Optimal Correspondence

PENGFEI XU, Shenzhen University, China
YIFAN LI, Shenzhen University, China
ZHIJIN YANG, Shenzhen University, China
WEIRAN SHI, Shenzhen University, China
HONGBO FU, City University of Hong Kong, China
HUI HUANG*, Shenzhen University, China

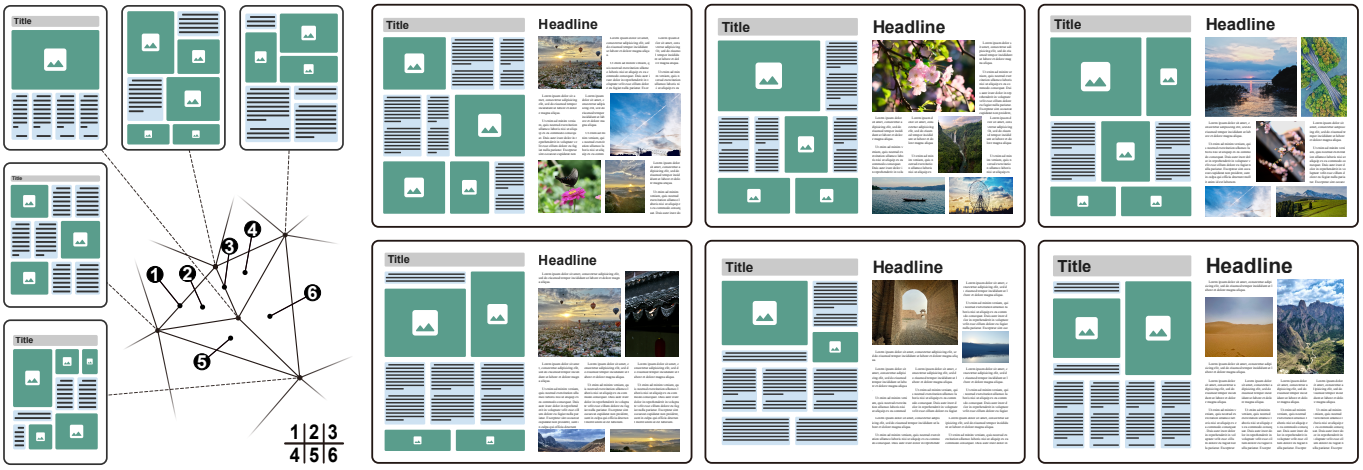

Fig. 1. Our hierarchical layout blending method can produce novel layouts (right) with given ones (left). Our method considers the semantic information of the input layouts, and thus can generate realistic layouts with reasonable structures. The generated layouts can be used for visual content creation, e.g., magazine page creation. The structural similarity measure introduced by our method can be used to construct a layout space (middle bottom), which can be integrated with machine/deep learning techniques for tasks like sketch-based layout retrieval. Image credits: ©Weiran Shi.

We present a novel method for blending hierarchical layouts with semantic labels. The core of our method is a hierarchical structure correspondence algorithm, which recursively finds optimal substructure correspondences, achieving a globally optimal correspondence between a pair of hierarchical layouts. This correspondence is consistent with the structures of both layouts, allowing us to define the union of the layouts' structures. The resulting compound structure helps extract intermediate layout structures, from which blended layouts can be generated via an optimization approach. The correspondence also defines a similarity measure between layouts in a hierarchically structured view. Our method provides a new way for novel layout creation. The introduced structural similarity measure regularizes the layouts in a hyperspace. We demonstrate two applications in this paper, i.e., exploratory design of novel layouts and sketch-based layout retrieval, and test them on a magazine layout dataset. The effectiveness and feasibility of these two applications are confirmed by the user feedback and the extensive results. The code is available at https://github.com/lyf7115/LayoutBlending.

CCS Concepts: • **Computing methodologies → Computer graphics**.

Additional Key Words and Phrases: layout blending, layout creation, hierarchical structure correspondence, layout retrieval

ACM Reference Format:
Pengfei Xu, Yifan Li, Zhijin Yang, Weiran Shi, Hongbo Fu, and Hui Huang. 2022. Hierarchical Layout Blending with Recursive Optimal Correspondence. *ACM Trans. Graph.* 41, 6, Article 249 (December 2022), 15 pages. https://doi.org/10.1145/3550454.3555446

*Corresponding author.

Authors' addresses: Pengfei Xu, xupengfei.cg@gmail.com, College of Computer Science & Software Engineering, Shenzhen University, China; Yifan Li, liyifan7115@gmail.com, College of Computer Science & Software Engineering, Shenzhen University, China; Zhijin Yang, yangzhijin1998@gmail.com, College of Computer Science & Software Engineering, Shenzhen University, China; Weiran Shi, weiran.better@gmail.com, College of Computer Science & Software Engineering, Shenzhen University, China; Hongbo Fu, hongbofu@cityu.edu.hk, School of Creative Media, City University of Hong Kong, China; Hui Huang, hhzhiyan@gmail.com, College of Computer Science & Software Engineering, Shenzhen University, China.

0730-0301/2022/12-ART249 $15.00
https://doi.org/10.1145/3550454.3555446

1 INTRODUCTION

The design of novel layouts is a fundamental problem for visual content creation and has received persistent interests from the research community. Some studies focus on facilitating interactive layout creation with intelligent tools [Baudisch et al. 2005; Bier and Stone 1986; Raisamo and Räihä 1996; Xu et al. 2015]. Designing complex layouts with these tools requires professional skills and

heavy labor input of users. Recently, advanced learning techniques have benefited automatic creation of layouts. A large number of generative models (e.g., [Gupta et al. 2021; Hu et al. 2020; Jyothi et al. 2019; Kikuchi et al. 2021; Lee et al. 2020; Li et al. 2019b; Patil et al. 2020; Tabata et al. 2019; Wu et al. 2019; Zheng et al. 2019]) have been recently proposed for the automatic creation of layouts. However, the requirement of a large dataset of layouts for training usually impedes their application in certain scenarios. Users may want to create variations from a small number of layouts in certain styles without resorting to heavy data collection and learning-based model training. For example, a common layout design procedure of posters or magazines would be first collecting a few samples of interest and then drawing inspiration from them [Kang et al. 2021].

Blending or interpolation is one of the fundamental means to create novel instances from a small number of instances. It generally requires first building a correspondence between every pair of instances. The intermediate instances can then be obtained with the guidance of the built correspondences. This technique has been used in different areas, such as inbeteweening [Reeves 1981], 2.5D cartoon models [Rivers et al. 2010], 2D sketch synthesis [Arora et al. 2017], 3D shape creation [Alhashim et al. 2014], view synthesis [Goesele et al. 2010], to name a few. Since this technique has benefited a wide range of instance creation problem, it seems promising to extend it to the novel layout creation task.

Layouts usually contain elements with parameterized representations. Intuitively, parameterized instances are usually suitable for blending, e.g, human face model [Vlasic et al. 2005] and human body model [Loper et al. 2015]. However, this is not the case for layouts. The main distinction of layouts from the above two parameterized models is that layouts' parameterized representations and structures are mutable and do not provide direct clues for the correspondence, which is crucial for the blending. For instance, layouts may contain different numbers of elements with various semantic labels and arrangements. In contrast, the face and body models have fixed representations and structures, leading to a fixed correspondence between different instances. Therefore, estimating an appropriate correspondence between layouts is necessary before blending them.

Due to the structural nature of layouts, we need to estimate the correspondence between layouts in the element level. In addition, to achieve the gradual blending of layout structures, this correspondence should be consistent with the structures of the input layouts, i.e., the corresponded elements should possess similar structures in different layouts. There exists algorithms [Kumar et al. 2011; Patil et al. 2020, 2021] for estimating correspondence between layouts. However, these algorithms fail to guarantee the consistency and thus are incapable of structure blending (please refer to Section 4 for illustration). Other element-level correspondence estimation algorithms [Alhashim et al. 2015; Zhu et al. 2017] were proposed for the application of 3D shape blending [Alhashim et al. 2014]. These algorithms respect the structure consistency, but they cannot be extended to layouts due to the following reasons. First, the geometry information of a layout is highly limited. Since a layout is usually composed of simple primitives, e.g., rectangles, its geometric features only provide limited hints for correspondence. Second and more importantly, layouts usually have complex and diverse hierarchical structures. Estimating an optimal hierarchical correspondence between a pair of layouts is challenging. Zhu et al. [2017] have attempted to find a near-optimal binary hierarchical correspondence between a pair of 3D models. However, their algorithm requires the input models to be in the same category, implicitly requiring similar structures of these models. In addition, their correspondence quality is evaluated via a data-driven criterion, which is difficult to extend to layouts. These demands restrict the extension of their algorithm to the layout blending problem.

In this paper, we present a novel method for blending hierarchical layouts with semantic labels. Our current implementation focuses on grid layouts, composed of rectangular semantic elements with regular arrangements. However, we believe our method is applicable to other layouts with properly defined arrangements, e.g., inforgraphic layouts or interior layouts. We do not consider layouts with irregular arrangements since blending them is trivial: any correspondence is applicable for producing blended layouts with such layouts, since the geometry of each element can be interpolated individually (see Figure 7).

Our method takes as input a pair of layouts with hierarchical structures represented as trees (see Figure 2). The hierarchical structures are obtained with geometric cues using a similar method described in [Jiang et al. 2021]. It first estimates an optimal correspondence between the input layouts by recursively finding optimal substructure correspondences. For any given pair of layouts, this strategy guarantees to find an optimal correspondence, which is consistent with the structures of both layouts. Along with the optimal correspondence, a correspondence cost is also returned, which reflects the similarity between the input layouts in a hierarchically structured view. After obtaining the optimal consistent correspondence, our method constructs a compound tree for embedding the structures of the input layouts. Given a blending factor, the intermediate structure of a blended layout can be determined from this compound tree. The blended layout is then generated based on its intermediate structure by an optimization approach.

The capacity of our method for blending layouts provides a new way for novel layout creation. The introduced structural similarity measure regularizes the layouts in a hyperspace and thus can be potentially used for constructing a layout space. Our method thus benefits various applications and we demonstrate two applications in this paper, tested on a dataset of magazine layouts [Zheng et al. 2019]. The first one is exploratory design of novel layouts by browsing blended layouts with user selected layouts. We design an interface for real-time displaying of blended layouts from input layouts with a navigation operation. We also design two local control tools to achieve asynchronous blending of layouts. The effectiveness of this application is confirmed by the user feedback. The second application is sketch-based layout retrieval. To achieve this, we construct a latent space of layouts using the structural similarity measure introduced by our method and embed line drawings of layouts in this latent space via a deep neural network. Given a hand-drawn sketch of a layout, the trained network is able to return its coordinate in the latent space, which can then be used for the layout retrieval. The tested results confirm the feasibility of this solution for sketch-based layout retrieval.

2 RELATED WORK

Layout creation. Layout creation is a fundamental problem for visual content design. Early research studies have mainly focused on facilitating interactive layout creation. Some of them have already been integrated in the mainstream graphic design frameworks as standard components, e.g., arrangement commands [Raisamo and Räihä 1996], snapping tools [Baudisch et al. 2005; Bier and Stone 1986]. Some more advanced techniques [Badros et al. 2001; Borning et al. 1997; Jiang et al. 2019; Xu et al. 2019] are able to regularize layouts by enforcing constraints between elements. With these techniques, users may avoid precise element placement, but still need to perform manual operations to specify desired layouts. In contrast, our method does not need manual operations for layout creation. Recently, due to the development of machine/deep learning, a lot of works (e.g., [Gupta et al. 2021; Hu et al. 2020; Jyothi et al. 2019; Kikuchi et al. 2021; Lee et al. 2020; Li et al. 2019b; Patil et al. 2020; Tabata et al. 2019; Wu et al. 2019; Zheng et al. 2019]) solve the layout creation/generation problem in a data-driven manner. These works are able to produce layouts automatically, with user inputs to specify constraint features of desired layouts if necessary. While these works need the support of a large dataset of layouts for training, our method needs only a small set of exemplars to create novel layouts.

Content creation through blending. Blending or interpolation provides an effective way to create novel instances from a small number of given ones. It has been adopted in a lot of areas. In the following we describe one representative work for each area. Reeves [1981] presented an early work about the inbetweening for computer animation and emphasized the importance of correspondence establishment. Rivers et al. [2010] proposed a 2.5D cartoon model for synthesizing in-between cartoon objects and characters in novel views. This was achieved by interpolating given cartoon images. Arora et al. [2017] presented a workflow that allowed designers to explore the design space induced by given sketches. The novel sketches in this design space were created using interpolation. Alhashim et al. [2014] introduced an algorithm for creating novel 3D models via structural blending. In their work, a 3D model was represented as a graph without hierarchy. Goesele et al. [2010] presented an image-based view interpolation algorithm with the assistance of ambient point clouds. For part-based or component-based instances (e.g., many man-made models), assembly-based or combination-based synthesis methods [Jain et al. 2012; Kalogerakis et al. 2012] also achieve blending effects of multiple instances. All the aforementioned methods require correspondence between instances before using the blending techniques. Our method also needs to estimate a correspondence between layouts before layout blending. The challenge is that layouts usually possess hierarchical structures [Stuerzlinger et al. 2006], which introduce extra difficulties to the estimation of the correspondence. There were some attempts on the layout blending problem. Bao et al. [2013] presented a framework for locally blending layouts with the same structures. Jiang et al. [2021] presented a framework for blending UI layouts with different structures. However, their framework requires a ground-truth correspondence as input. In contrast, our method is able to automatically estimate a hierarchical structure correspondence between layouts for blending.

Element-level correspondence. There exist several algorithms for estimating correspondence between layouts. Kumar et al. [2011] presented *Bricolage* for web design retargeting. This is achieved by estimating the correspondence between hierarchical layouts with a probabilistic approach. Patil et al. [2020] presented a recursive neural network for document layout generation. Their layout similarity measure requires finding a correspondence between layouts. *LayoutGMN* [Patil et al. 2021] is a neural network for predicting structural similarity between layouts. The learned attention weights in this network can be used to match the elements in layouts. However, none of these methods guarantees to produce correspondences that are consistent with the structures of layouts, and thus they cannot be applied to layout blending.

An appropriate correspondence is the prerequisite of interpolation or blending. The application of example-driven 3D model deformation or morphing requires estimating dense correspondence between a pair of 3D models [Huang et al. 2008]. Recently, more efforts have been put on the part-level correspondence, enabling higher-level manipulation of shapes. Alhashim et al. [2015] adopted a deformation energy to estimate part-level correspondence between 3D shapes with different structures. Zhu et al. [2017] presented a method for estimating part-level correspondence between 3D shapes using a data-driven plausibility criterion. Both methods assume that the input shapes are in the same category, implicitly requiring similar structures of these models. In contrast, our method does not have such requirements on the structures of the input layouts.

Our method adopts a recursive strategy to find the correspondence between two layouts. This strategy has also been used in existing algorithms [Schwarz et al. 2017; Zhang and Shasha 1989] for estimating correspondences and defining distances between tree structures. Compared with our method, these algorithms have different problem settings. For example, they require the input tree structures to be ordered while our method does not. In addition, they cannot enforce the correspondence rules (Section 3.2) that we conclude for the layout blending. Therefore, these algorithms are not applicable to the layout blending problem.

Layout similarity. For a long period, the similarity measure of layouts relies heavily on segmentation metrics such as intersection-over-union (IoU) [Hu et al. 2020; Manandhar et al. 2020; Patil et al. 2021] and human judgement [Hu et al. 2020; Li et al. 2019a]. The structural information of layouts is often ignored. Recently, some works [Manandhar et al. 2020; Patil et al. 2021] tried to define similarity measures between layouts using graph neural networks [Scarselli et al. 2008]. They adopted a triplet loss for training, with positive and negative samples determined by the IoU measure. Although these works define reasonable similarity measures on layouts, the hierarchical structures of the layouts are not considered in the training. In contrast, our layout similarity relies on estimating a hierarchical structure correspondence, and thus is a hierarchical structure-aware layout similarity measure.

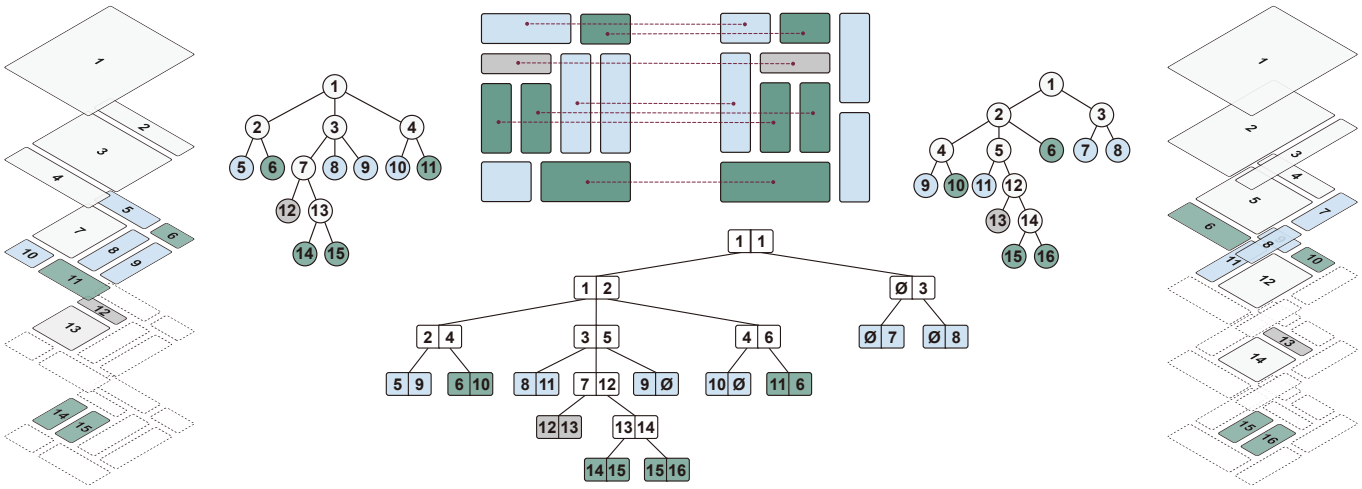

Fig. 2. The illustration of how the hierarchical structure correspondence is estimated. Left and right: two input layouts with hierarchical structures represented as layout trees. Top middle: the correspondence between the elements in the two layouts (i.e., the leaf vertices in the two layout trees). The colors of the elements represent different semantic labels. Bottom middle: the constructed compound tree.

3 METHOD

To achieve the blending of layouts with hierarchical structures, the most fundamental problem is to estimate the correspondence between two layouts. We first introduce the representation of layouts in our method (Section 3.1) and then conclude several rules for the hierarchical structure correspondence (Section 3.2). Guided by these rules, we design an algorithm to find a globally optimal correspondence by recursively estimating the optimal correspondences of substructures (Section 3.3). With the estimated correspondence, we can construct a compound structure and extract an intermediate layout structure (Section 3.4). This structure helps generate the final layout using an optimization approach (Section 3.4).

3.1 Layout representation

The input of our layout blending method is a pair of 2D structural layouts, which are composed of parameterized rectangular elements and have hierarchical structures. In practice, the extraction of a hierarchical structure of a layout usually depends on predefined rules, e.g., semantic or geometric cues, which may vary in different scenarios. Our method solves the layout blending problem in a geometric way, thus requiring the hierarchical structures to be extracted from geometric cues. In our current implementation, we define the hierarchy of a layout similar to [Jiang et al. 2021], i.e., elements are grouped if they form a horizontal or vertical arrangement, which means these elements are placed consecutively in a horizontal or vertical order. The grouped elements are then considered as a new entity, which will be further grouped with other entities according to the same geometric cues. This procedure iterates until all the elements are processed. In our implementation, the hierarchy is obtained by recursively splitting the layouts with horizontal or vertical lines that do not traverse elements. Please refer to Figure 2 for the hierarchy construction. We adopt this grouping strategy mainly due to its simplicity and might adopt other clustering methods[Koch and Oulasvirta 2016], as long as the criteria are based on geometric information. To capture the hierarchical structure of a layout, we adopt a layout tree [Jiang et al. 2021] $\mathcal{T} = \{\mathcal{V}, \mathcal{E}\}$ as its representation, instead of a fully connected graph [Patil et al. 2021], which treats all elements equally. In this layout tree (see Figure 2), each leaf vertex represents an element in an input layout and each branch vertex represents an arrangement of its children. The edges indicate the hierarchical relations between vertices.

3.2 Correspondence rules

Given two input layouts with the computed hierarchical structures, i.e., $\mathcal{T} = \{\mathcal{V}, \mathcal{E}\}$ and $\mathcal{T}' = \{\mathcal{V}', \mathcal{E}'\}$, we need to find a reasonable correspondence between them. Note that the correspondence should not only consider the leaf vertices of the two corresponding layout trees, i.e., the elements of the two layouts, but also take into account their branch vertices, i.e., the arrangements of elements. We observe a desired correspondence should fulfill the following requirements. First, the correspondence should consider the geometry of elements, e.g., a pair of elements with similar shapes and locations in two layouts tend to correspond to each other. Second and more importantly, the correspondence should respect the hierarchical structures of the layouts. The structures should be consistent with each other under the correspondence. Otherwise, it is not possible to perform a smooth structure blending and the blended layouts would have abrupt change of the topologies (see Figure 6). The first requirement can be easily achieved by adopting an appropriate deformation cost [Kleiman et al. 2015], while the second requirement is nontrivial to enforce. Before finding the optimal correspondence, we first identify some rules (see Figure 3) based on the above requirements.

Rule 1. Two leaf vertices in two layout trees with similar geometry tend to correspond to each other. Two branch vertices with

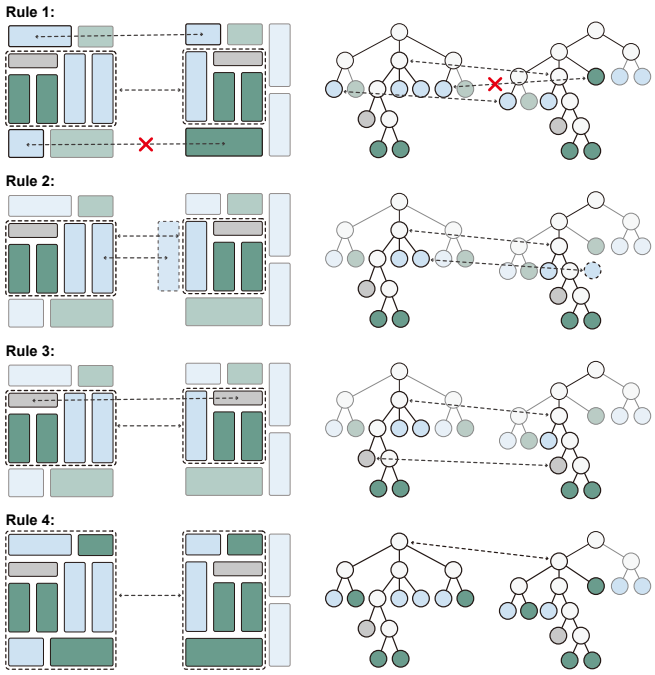

Fig. 3. The illustration of the rules for hierarchical structure correspondence. The colors indicate the semantic labels of the elements (leaf vertices). The dashed frames highlight arrangements of elements, which are represented by branch vertices in layout trees. Please refer to Section 3.2 for a more detailed description about the rules.

similar substructures and geometry tend to correspond to each other. Two leaf vertices with different semantic labels are not allowed to correspond to each other.

Rule 2. A vertex v, which may be a leaf vertex or a branch vertex, is allowed to correspond to a void $\emptyset$. This is essential since two layouts usually contain different numbers of elements and it is not always possible to form a one-to-one mapping. In addition, even if two layouts contain an equal number of elements, they may be distinct in hierarchical structure and/or geometry of elements. In this case, a one-to-one mapping is usually not preferable.

Rule 3. If a vertex $v \in \mathcal{T}$ corresponds to a vertex $v' \in \mathcal{T}'$, then the vertices in the subtree $\mathcal{T}_{sub}(v)$ rooted at vertex v can only correspond to the vertices in the subtree $\mathcal{T}'_{sub}(v')$ or voids, and vice versa. This rule is important for keeping the hierarchical structures consistent during the blending of two layouts.

Rule 4. Vertices with different levels in two layout trees are allowed to correspond. This rule is based on the fact that two similar local arrangement patterns may have different hierarchical levels in two layouts.

3.3 Optimal hierarchical structure correspondence

The above rules describe the basic characteristics of a desired correspondence. It is still unclear how to obtain such correspondences. The first part of Rule 1, i.e., the correspondence between two leaf vertices, can be achieved by a deformation cost [Kleiman et al. 2015]. To successfully enforce all the rules, we adopt a recursive strategy to define the following correspondence costs.

Leaf-to-leaf cost. Guided by Rule 1, if two elements v and v' in two layouts have similar geometry, i.e., positions and shapes, the correspondence cost should be small. If they have different semantic labels, the cost should be infinity to prevent them from corresponding. We then define this cost with the following formulation:

$$C_{l-l}(v, v') = \left\|\mathbf{p}_v - \mathbf{p}_{v'}\right\|_2 + \left\|\mathbf{s}_v - \mathbf{s}_{v'}\right\|_2 + C_{label}(v, v'), \tag{1}$$

with

$$C_{label}(v, v') = \begin{cases} 0, & \text{if } v \text{ and } v' \text{ have same labels} \\ +\infty, & \text{otherwise} \end{cases}, \tag{2}$$

where $\mathbf{p}_v = (x_v, y_v)^T$ is the center of an element v, and $\mathbf{s}_v = (w_v, h_v)^T$ is its size. We adopt the commonly used L_2 norm for the cost computation. Other norms (e.g., L_1 norm) may also apply. C_{label} is a label-difference cost, which penalizes the correspondence between elements with different labels. In our implementation, we replace $+\infty$ with a large number (e.g., 10^9). This cost is a basic one without considering any structure information.

Leaf-to-void cost. According to Rule 2, a leaf vertex is allowed to correspond to a void. Note that this leaf vertex can be in both input layout trees (e.g., v_9 in the left tree and v'_7 in the right tree in Figure 2). Similar to the leaf-to-leaf cost, we define this cost as the effort for deforming an element v to a void. For a void element, it is reasonable to consider its size as zero. However, there is no hint to define its position. We add an extra penalty cost $C_\emptyset$ to compensate the position deformation cost. Without this compensation, an element might more tend to correspond to a void. Then this leaf-to-void cost is defined as:

$$C_{l-\emptyset}(v) = \left\|\mathbf{s}_v\right\|_2 + C_\emptyset, \tag{3}$$

where $C_\emptyset = \left\|\mathbf{s}_v/2\right\|_2$ in our implementation. This cost is equivalent to a position deformation cost by moving an element with a distance same as its half size.$C_\emptyset$ controls the priority of the correspondence between an element and a void. If $C_\emptyset$ is small, an element is more likely to correspond to a void (as opposed to an element), even if it could have corresponded to another element. If $C_\emptyset$ is large, an element is more likely to correspond to another element (as opposed to a void), even if this pair of corresponded elements have a large location difference. The leaf-to-void cost is another basic cost without considering the structure information.

Branch-to-void cost. A branch vertex represents an arrangement of its children, which may be elements or lower-level arrangements. Deforming an arrangement to a void is equivalent to deforming all the elements in this arrangement to voids (e.g., v'_3 in the right tree in Figure 2). Therefore, this cost is defined as:

$$C_{b-\emptyset}(v) = \sum_{v_i} C_{l-\emptyset}(v_i), \tag{4}$$

where v_i is a leaf descendant of v, i.e., an element contained in this arrangement.

Branch-to-leaf cost. This cost is based on the deformation between an arrangement and an element (e.g., v_4 in the left tree and v'_6 in the right tree in Figure 2). Note that this deformation may include emergence and vanish of elements (e.g., v_{10} in the left tree in Figure 2). It is not equivalent to the geometric difference between the bounding boxes of the arrangement and the element. Guided by Rules 3 and 4, we adopt a recursive way to define their correspondence cost.

We consider the case in which v is a branch vertex and v' is a leaf vertex (e.g., v_4 in the left tree and v'_6 in the right tree in Figure 2). The computation for the symmetric case is similar. We first compute the optimal correspondence between the children of v and v'. Without loss of generality, we assume that the correspondence cost between any child of v and v' is available. This assumption is reasonable since these costs can be obtained recursively with the previously defined cost functions. Then the optimal correspondence can be obtained in polynomial time using the Hungarian algorithm [Kuhn 1955] to be described later. If we denote the optimal correspondence as a pair set $\mathcal{P} = \{(v_i, v'_j)\}$, where v_i is a child of v or a void, v'_j is v' or a void, then the branch-to-leaf cost can be defined as:

$$C_{b-l}(v, v') = \sum_{(v_i, v'_j) \in \mathcal{P}} C_*(v_i, v'_j), \tag{5}$$

where C_* is the appropriate cost function determined by the types of v_i and v'_j, i.e., if v_i is a branch vertex, then $C_*(v_i, v'_j) = C_{b-l}(v_i, v'_j)$; if v_i is a void, then $C_*(v_i, v'_j) = C_{l-\emptyset}(v'_j)$, etc. Note that, the children of v and v' may all correspond to voids, e.g., the children of v and v' may have different labels.

Branch-to-branch cost. This cost is based on the deformation between two arrangements v and v' (e.g., v_1 in the left tree and v'_2 in the right tree in Figure 2). This deformation involves both the evolution of the structure and the change of the elements' geometry. Similar to the branch-to-leaf cost, we also adopt a recursive way to define this cost. Since both v and v' have children, the optimal correspondence may be obtained in one of the following three configurations: a), the children of v correspond to the children of v'; b), the children of v correspond to v'; and c), v corresponds to the children of v'. Denoting the optimal correspondences for these three configurations as $\mathcal{P}_a$, $\mathcal{P}_b$, and $\mathcal{P}_c$, we define the branch-to-branch cost as:

$$C_{b-b}(v, v') = \min\{C_\diamond(v, v')\}_{\diamond \in \{a,b,c\}}, \tag{6}$$

with

$$\begin{aligned} C_\diamond(v, v') &= \sum_{(v_i, v'_j) \in \mathcal{P}_\diamond} C_*(v_i, v'_j), \\ \diamond &\in \{a, b, c\}, \end{aligned} \tag{7}$$

where C_* is the appropriate cost function determined by the types of v_i and v'_j, i.e., if v_i and v'_j are branch vertices, $C_* = C_{b-b}$; if v_i is a leaf vertex and v'_j is a branch vertex, $C_* = C_{b-l}$, etc.

Hungarian algorithm. The definitions of the branch-to-branch cost and the branch-to-leaf cost require finding the optimal correspondence between two sets of vertices $\{v_i\}_{i=1}^m$ and $\{v'_j\}_{j=1}^n$. Here we assume that the correspondence costs for pair (v_i, v'_j), $(v_i, \emptyset)$, and $(\emptyset, v'_j)$ are available since they can be obtained recursively with the previously defined cost functions. We then use the Hungarian algorithm [Kuhn 1955] to solve this correspondence problem. We first construct the following cost matrix $\mathbf{M}$:

$$\mathbf{M} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{0} \end{pmatrix}, \tag{8}$$

where $\mathbf{A}$ is an $m \times n$ matrix with $\mathbf{A}_{ij} = C_*(v_i, v'_j)$, which encodes the correspondence cost between v_i and v'_j; $\mathbf{B}$ is an $m \times m$ matrix with $\mathbf{B}_{ij} = C_{*-\emptyset}(v_i)$, which encodes the correspondence cost between v_i and a void; $\mathbf{C}$ is an $n \times n$ matrix with $\mathbf{C}_{ij} = C_{*-\emptyset}(v'_j)$, which encodes the correspondence cost between v'_j and a void. C_* inherits the previous definition; $C_{*-\emptyset}$ is $C_{l-\emptyset}$ or $C_{b-\emptyset}$, depending on the type of the parameter. Solving this correspondence problem using the Hungarian algorithm gives the optimal correspondence.

Global optimal correspondence. Given two input layouts $\mathcal{T}$ and $\mathcal{T}'$, their optimal correspondence can be obtained by computing the correspondence cost between their root vertices. The correspondence of the vertices in $\mathcal{T}$ and $\mathcal{T}'$ is determined recursively. In this procedure, the correspondence cost between any pair of vertices is computed based on an optimal correspondence, and thus the final correspondence between the input layouts is globally optimal. The correspondence cost between the root vertices reflects the difference of the input layouts, and thus is suitable as a similarity measure.

Effect of the deformation cost. We adopt a simple definition of the deformation cost for the leaf-to-leaf cost. This is due to the simple geometry of a rectangle. The position deformation cost $\left\|\mathbf{p}_v - \mathbf{p}_{v'}\right\|_2$ and the size deformation cost $\left\|\mathbf{s}_v - \mathbf{s}_{v'}\right\|_2$ have the same contribution to the leaf-to-leaf cost. We may add weights to control their effects on the correspondence. For example, if $\left\|\mathbf{s}_v - \mathbf{s}_{v'}\right\|_2$ has a larger weight, then elements with similar sizes will tend to correspond, even they have distant locations. In all of our experiments, we treat these two costs equally.

Binary deformation cost. We only adopt unary deformation costs in our method. We do not explicitly define a binary deformation cost [Kleiman et al. 2015], which encodes pairwise relations between elements, e.g., the alignment relations between elements. It is reasonable to abandon this kind of deformation costs since the pairwise relations, even group relations of elements in a layout, have already been encoded in a hierarchical structure.

Computational complexity. To obtain the final correspondence, our method needs to compute the correspondence costs of all pairs of vertices in two input layout trees. For each pair of vertices, the computation time depends on the vertices' types. In the worst case, it needs to solve a matching problem with the Hungarian algorithm, whose computational complexity is $O(N^3)$, where N is the number of instances being corresponded. If the vertex sets of two layout trees are $\mathcal{V}$ and $\mathcal{V}'$, and the maximum number of children vertices of a vertex in the layout trees is K, then the computational complexity is $O(|\mathcal{V}| \cdot |\mathcal{V}'| \cdot K^3)$. Note that this bound is quite loose: the matching procedure with the Hungarian algorithm is not necessary for most computations of correspondence costs between vertices. In practice, typically, the number of elements in a layout is smaller than hundreds and the maximum number of children vertices of a vertex is smaller than dozens. Therefore our method is extremely

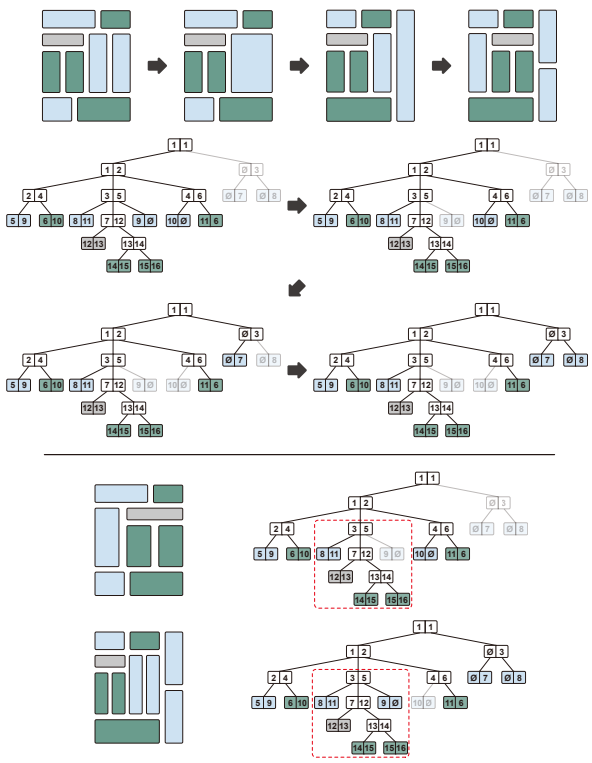

Fig. 4. Top: an interpolation procedure between two input layouts. Middle: the corresponding intermediate trees in each stage of the interpolation. Bottom: two blended layouts with asynchronous interpolations. The removed vertices are rendered in a transparent style. The local editing is highlighted with red dashed frames. Row 4: the global α is 0 and the local α is 1. Row 5: the global α is 1 and the local α is 0.

efficient for processing layouts. With our C++ implementation running on an ordinary PC, it takes less than 2 milliseconds to compute a correspondence between two layouts presented in this paper.

3.4 Layout blending

After obtaining the correspondence between the input layouts, we can create novel layouts via blending. This is achieved with three steps. First, we construct a compound tree, which is the union of the input layout trees. We then extract an intermediate layout tree with a given blending factor. Finally, we can produce a blended layout from the intermediate layout tree with an optimization approach.

Compound tree. The structures of the input layouts are consistent with the correspondence obtained by our method. That is, if (v_i, v'_j) and (v_p, v'_q) are two pairs of corresponded vertices, where $v_i, v_p \in \mathcal{T}$ and $v'_j, v'_q \in \mathcal{T}'$, we can conclude that v_i is a descendant (or ancestor) of v_p if and only if v'_j is a descendant (or ancestor) of v'_q. This is implied by Rule 3. Therefore, it is possible to construct a compound tree by uniting the input layout trees without destroying their structures (see Figure 2). A vertex of this compound tree is a pair (v_i, v'_j), where $v_i \in \mathcal{T} \cup \{\emptyset\}$ and $v'_j \in \mathcal{T}' \cup \{\emptyset\}$. Therefore, the vertices of this compound tree can be classified into three types. The first type of vertices indicate pairs of corresponded vertices in $\mathcal{T}$ and $\mathcal{T}'$. For the other two types, either v_i or v'_j is a void $\emptyset$.

Intermediate layout tree extraction. An intermediate layout tree is a subtree of the compound tree. It can be extracted by properly removing the vertices of the compound tree, and this removal procedure is described as follows. Note that only the last two types of vertices, i.e., either v_i or v'_j is a void $\emptyset$, are removable. This removal can be performed with different strategies, with respect to a blending factor $\alpha \in [0, 1]$. Intuitively, the blended structure is the same as $\mathcal{T}$ or $\mathcal{T}'$ if $\alpha = 0$ or 1. This can be achieved by removing the vertices that are in the form of $(\emptyset, v'_j)$ with $v'_j \in \mathcal{T}'$ (if $\alpha = 0$), or $(v_i, \emptyset)$ with $v_i \in \mathcal{T}$ (if $\alpha = 1$), respectively. To obtain a smooth transition between $\mathcal{T}$ and $\mathcal{T}'$, we may gradually remove vertices $(v_i, \emptyset)$ and $(\emptyset, v'_j)$ according to the blending factor α (see Figure 4). A simple strategy is to remove the leaf vertices that are in the form of $(v_i, \emptyset)$ and $(\emptyset, v'_j)$ one by one from the lowest level to the highest level. We denote the extracted intermediate layout tree with the blending factor α as $\mathcal{T}^\alpha$ and its vertices are $\{v^\alpha_k = (v_i, v'_j)\}$. Different strategies for the interpolation may also apply, which will result in alternative paths for the transition. For example, in Section 5.1, we allow the blending factors to be different in different parts of the compound tree (Figure 4, bottom), achieving an asynchronous interpolation effect. If there is no vertex in the form of $(\emptyset, v'_j)$ or $(v_i, \emptyset)$ in the compound tree, it implies that two input layouts have identical structures and all their elements are corresponded. In this case, only the geometry of elements needs to be blended.

Layout generation using optimization. After extracting the intermediate layout tree, the final layout can be easily obtained with existing constraint-based layout optimization approaches [Badros et al. 2001; Jiang et al. 2020; Xu et al. 2019]. We adopt the approach proposed in [Xu et al. 2019] since it optimizes layouts that are in similar forms to ours. Specifically, for preparation, we store the relations among the children vertices at each vertex $v_i \in \mathcal{T}$ or $v'_j \in \mathcal{T}'$. In our implementation, we consider alignment, equal-spacing, and equal-size relations only. More advanced constraints such as symmetry or containing might also be included for specific applications. In the intermediate layout tree, a vertex (v_i, v'_j) may correspond to elements in both $\mathcal{T}$ and $\mathcal{T}'$, introducing ambiguity for the relation selection. We adopt a simple strategy to solve this problem. For the relation between (v_i, v'_j) and (v_p, v'_q), the selection is based on the blending factor α, i.e., if $\alpha < 0.5$, the relation is the same as the one between v_i and v_p; otherwise, it is determined by the relation between v'_j and v'_q.

After determining the relations among the vertices in the intermediate layout tree $\mathcal{T}^\alpha$, we are ready to compute the final layout. Similar to [Xu et al. 2019], we define an objective function as follows:

$$E = E_{\mathbf{p}} + E_{\mathbf{s}}, \tag{9}$$

with

$$\begin{aligned} E_{\mathbf{p}} &= \sum_{v^\alpha_k} \left((1-\alpha)\delta(v_i) \left\| \mathbf{p}_{v^\alpha_k} - \mathbf{p}_{v_i} \right\|^2 + \alpha\delta(v'_j) \left\| \mathbf{p}_{v^\alpha_k} - \mathbf{p}_{v'_j} \right\|^2 \right), \\ E_{\mathbf{s}} &= \sum_{v^\alpha_k} \left((1-\alpha) \left\| \mathbf{s}_{v^\alpha_k} - \delta(v_i)\mathbf{s}_{v_i} \right\|^2 + \alpha \left\| \mathbf{s}_{v^\alpha_k} - \delta(v'_j)\mathbf{s}_{v'_j} \right\|^2 \right), \end{aligned} \tag{10}$$

where $v_k^\alpha = (v_i, v'_j)$ is a vertex in the intermediate tree $\mathcal{T}^\alpha$. $\mathbf{p}_{v_k^\alpha} = (x_{v_k^\alpha}, y_{v_k^\alpha})^T$ and $\mathbf{s}_{v_k^\alpha} = (w_{v_k^\alpha}, h_{v_k^\alpha})^T$ are the unknowns, representing the position and size of the vertex v_k^α. $\delta(v)$ is an indicator function whose value is 0 if v is a void $\emptyset$ and 1 otherwise. Minimizing the objective function is subject to several constraints, which are derived from the relations among the vertices, including alignment, equal-spacing, and equal-size. Another set of constraints are the bounding box constraints, i.e., the shape of a vertex should tightly envelop the shapes of its children vertices. The formulations of the alignment and equal-size constraints are similar to those in [Xu et al. 2019].For example, if v_k^α and v_l^α are vertically aligned, we have:

$$\begin{aligned} x_{v_k^\alpha} + \frac{1}{2} w_{v_k^\alpha} &= x_{v_l^\alpha} + \frac{1}{2} w_{v_l^\alpha}, \\ x_{v_k^\alpha} - \frac{1}{2} w_{v_k^\alpha} &= x_{v_l^\alpha} - \frac{1}{2} w_{v_l^\alpha}. \end{aligned} \tag{11}$$

The equal-spacing and bounding box constraints are dynamically determined and enforced with a two-step strategy. In the first step, we solve the optimization problem with alignment and equal-size constraints only. We then determine the equal-spacing and bounding box constraints according to the layouts generated in the first step. For instance, if v_k^α, v_l^α, and v_m^α are vertically aligned in a consecutive order, their equal-spacing constraint can be formulated as:

$$(y_{v_l^\alpha} - \frac{1}{2} h_{v_l^\alpha}) - (y_{v_k^\alpha} + \frac{1}{2} h_{v_k^\alpha}) = (y_{v_m^\alpha} - \frac{1}{2} h_{v_m^\alpha}) - (y_{v_l^\alpha} + \frac{1}{2} h_{v_l^\alpha}). \tag{12}$$

The bounding box constraints can be formulated similarly. Since these constraints are all linear, this optimization problem is a standard linear-equality-constrained quadratic programming problem, which can be solved efficiently. In our implementation, we solve it using the method of Lagrange multipliers with the Eigen library [Guennebaud et al. 2010]. The layout generation achieves a real-time performance (less than 20 milliseconds for each generation).

Blending multiple layouts. Our method can be easily extended to blending multiple layouts. We take the case of three layouts for illustration. Suppose the input layouts are $\mathcal{T}$, $\mathcal{T}'$, and $\mathcal{T}''$. We first compute the correspondence between two layouts, e.g., $\mathcal{T}$ and $\mathcal{T}'$, and obtain a compound tree $\widetilde{\mathcal{T}}$. Then the correspondence between $\widetilde{\mathcal{T}}$ and $\mathcal{T}''$ can be obtained using the same method. We only need to redefine the leaf-to-leaf cost and the leaf-to-void cost without modifying other costs, e.g., $\widetilde{C}_{l-l}((v_i, v'_j), v''_k) = C_*(v_i, v''_k) + C_*(v'_j, v''_k)$, where C_* is C_{l-l} or $C_{l-\emptyset}$, according to the types of the parameters. Based on this correspondence, we may construct a new compound tree whose vertices are triplets of vertices in $\mathcal{T}$, $\mathcal{T}'$, and $\mathcal{T}''$. We can then generate a blended layout with similar intermediate tree extraction and layout optimization approaches. Note that different orders of the correspondence computations may lead to different final compound trees, and thus may generate different layouts with the same blending factors. A simple strategy is to first compute the correspondence between layouts with a higher structural similarity since it may produce more compact compound trees. This extension of blending multiple layouts is used in one of our demonstrated applications, i.e., exploratory design of layouts (Section 5.1).

Blending unaligned layouts. Our method aims to blend structured layouts with regular arrangements. In practice, some layouts may present clear vertical or horizontal arrangements, but are not perfectly aligned (see two input layouts in Figure 5). This introduces challenges to the layout optimization due to the lack of appropriate alignment constraints. By adding extra padding elements (Figure 5, the gray elements), we can convert unaligned layouts into aligned ones, which can then be fed to our method for layout blending.

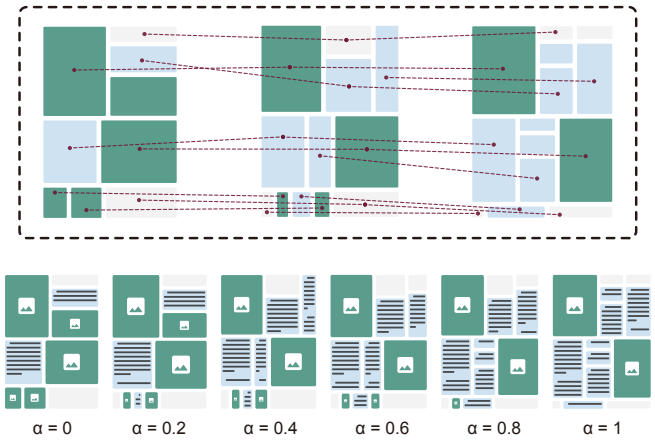

Fig. 5. Top: two input layouts (left and right) and a blended layout (middle). The gray elements are the padding elements, which make the layouts well aligned. The dashed lines indicate the correspondence between the layouts. Bottom: the blended layouts with the gradually changed blending factors.

4 EVALUATION

The core of hierarchical layout blending is the estimation of a correspondence that is consistent with the structures of the input layouts. To the best of our knowledge, there is no existing work trying to solve the hierarchical layout blending problem in this work. There exist several works [Kumar et al. 2011; Patil et al. 2020, 2021] that can estimate a correspondence between layouts. However, as discussed in Sections 1 and 2, they fail to guarantee the consistency of the correspondence and thus are incapable of layout blending. Besides geometric methods like ours, another possible direction for blending layouts is resorting to learning-based generative models [Kikuchi et al. 2021; Li et al. 2019b]. The layout blending could be achieved by first blending the latent codes of input layouts, and then generating a blended layout with the generative models. In this section, we conduct two comparisons with existing layout correspondence methods and a learning-based generative model respectively, and provide additional results to show the effectiveness of our method.

4.1 Validation of consistent correspondence

This comparison aims to illustrate the importance of an appropriate correspondence to the task of layout blending. We compare our method with *LayoutGMN* [Patil et al. 2021], *DocSim* [Patil et al. 2020], and maximum pixel-overlap matching (*Max-overlap* for short) [Patil et al. 2021]. *LayoutGMN* is a neural network for predicting structural similarity between layouts. The correspondence between two layouts can be estimated from the attention weights of the network. *DocSim* is a layout similarity measure which requires finding

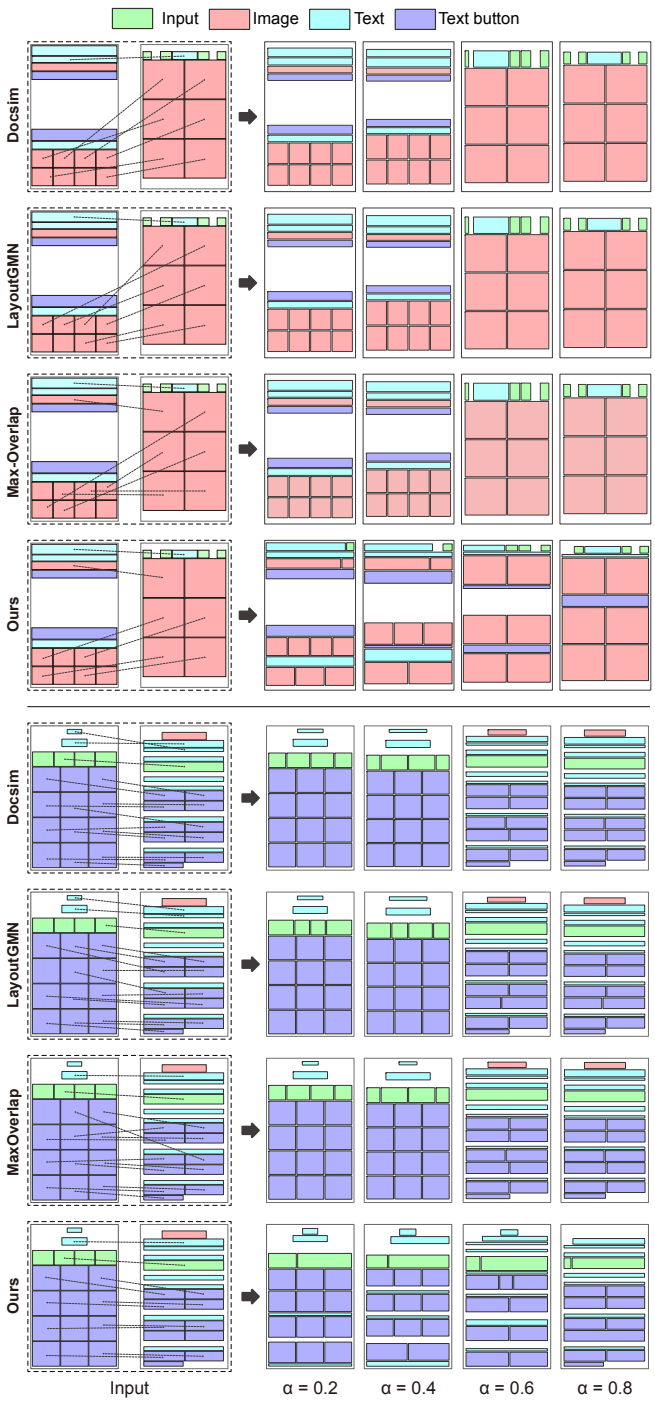

Fig. 6. The correspondences estimated by the compared methods and the blended layouts generated based on them. Layouts in this comparison are selected from the RICO dataset. The first two columns show the input layouts and the correspondences estimated by different methods; the last four columns show the blended layouts generated based on the estimated correspondences using our layout optimization method.

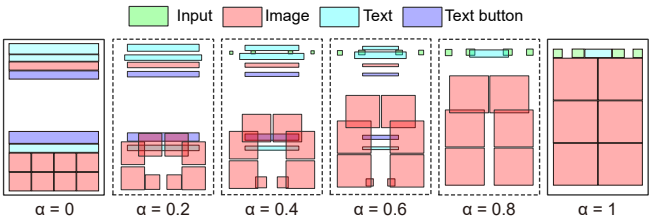

Fig. 7. The blended layouts generated by discarding the structure and interpolating the geometry of each individual element. The input layouts are the same as the top example in Figure 6. The correspondence is estimated by *DocSim* (Figure 6, row 1).

a correspondence between layouts. *Max-overlap* is a baseline correspondence method described in [Patil et al. 2021]. All these three methods produce element-level correspondence weight matrices. *DocSim* computes the final correspondence with the Hungarian algorithm. *LayoutGMN* estimates the final correspondence from the correspondence weight matrix (i.e., the attention weights) using a simple method: each element in one layout corresponds to the element with the largest attention weight in another layout. This method works well for the floorplan layouts provided in *LayoutGMN*. However, for the more complex layouts in our comparison, the Hungarian algorithm produces more reasonable correspondence results. Therefore, for a fair comparison, we adopt the Hungarian algorithm to estimate the optimal correspondence for *LayoutGMN* and *Max-overlap*. *LayoutGMN* is trained using the RICO dataset [Deka et al. 2017], which was initially adopted for the evaluation of *LayoutGMN*.

Figure 6 shows the correspondences estimated by the compared methods and the blended layouts generated based on them. Among all these correspondences, only the one estimated by our method is consistent with the structures of the input layouts. The blended layouts generated by our method demonstrate a smooth transition in both geometry and structure. Since the other methods cannot guarantee the consistency, it is impossible to perform the structure blending and intermediate structure extraction. If we simply discard the structure and blend the geometry of each individual element, the blended layouts will be unstructured and aesthetically undesirable (see Figure 7). To generate more reasonable blended layouts, we borrow one of the input layouts' structures for layout optimization. With the aid of our layout optimization method, the blended layouts are regular and appear smooth geometry transitions. However, they cannot achieve smooth structure transitions and there exists noticeable abrupt structure changes, as demonstrated in the accompanying video. The above comparison confirms the importance and effectiveness of our hierarchical structural correspondence method.

4.2 Comparison with generative model

There exist a number of generative models that are able to generate layouts. Some of them [Kikuchi et al. 2021; Li et al. 2019b; Patil et al. 2020] are able to embed layouts in a latent space and generate a novel layout with a latent code. A straightforward way for blending layouts with these methods is first blending the latent codes of input layouts and then generating a blended layout with the blended code. In this comparison, we compare our method with one of these

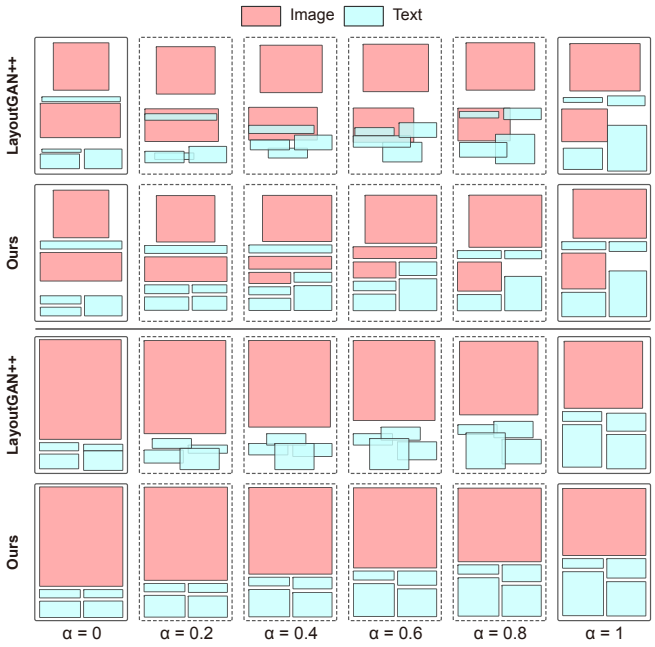

Fig. 8. The blended layouts generated by *LayoutGAN++* and our method. Our method achieves a smooth transition in both geometry and structure. *LayoutGAN++* only achieves a smooth geometry transition and the structures of its blended layouts are broken.

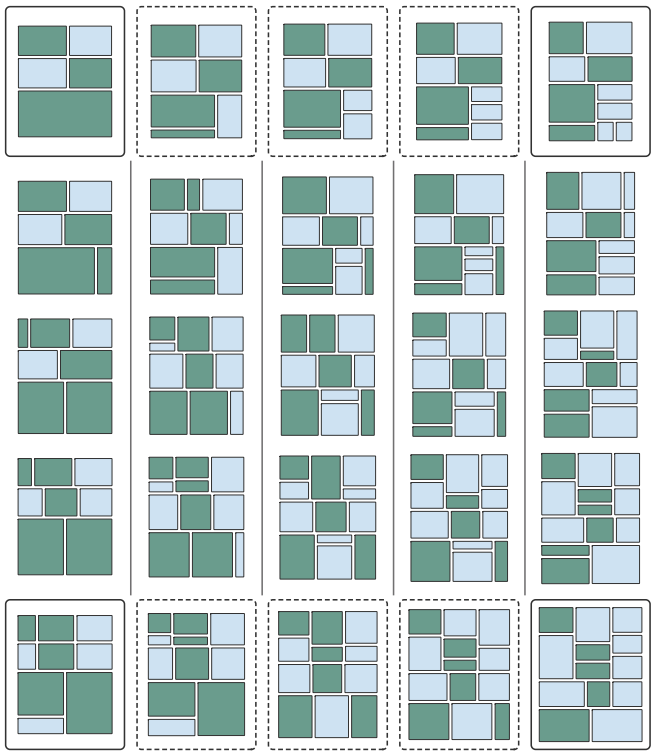

Fig. 9. Our method is able to produce layouts with smooth structure and geometry transitions. Four corners (highlighted by solid frames): four input layouts. Rows 1 and 5 (highlighted by dashed frames): the first-round blending between the input layouts at the corners of the same rows. Columns 1 to 5: the second-round blending between the blended layouts produced in the first round or between the input layouts in the same columns. The colors of the elements represent the semantic labels.

methods, i.e., *LayoutGAN++* [Kikuchi et al. 2021], which is the state of the art and open-sourced. In our comparison, *LayoutGAN++* is trained using the PubLayNet dataset [Zhong et al. 2019], which was initially adopted for its evaluation.

Figure 8 shows the layout blending results generated by our method and *LayoutGAN++*. Again, our method achieves a smooth transition in both geometry and structure. The blended layouts generated by *LayoutGAN++* present a smooth geometry transition, indicating the effectiveness of this generative model in layout creation. However, the structures of the blended layouts are broken. In addition, the underlying correspondence between the input layouts is not optimal, e.g., elements with distant locations are corresponded. Therefore, the blended layouts seem a little disordered. *LayoutGAN++* provides a function for layout beautification, which requires additional constraints as input. However, for the blended layouts, there is no clear hint to provide such constraints, since their structures are unknown. These results again confirm the importance of structure blending, which is the advantage of our method.

4.3 Additional results

We have tested our method with different types of layouts. Our method can generate plausible blended layouts (see Figure 11). Figure 9 shows additional results produced by our method. The blended layouts present a smooth transition in geometry and structure. In this example, we first blend two pairs of layouts (Figure 9, four corners) to generate two sets of blended layouts (Figure 9, rows 1 and 5). We then couple these layouts in the same columns for further blending, generating five new sets of blended layouts (Figure 9, columns 1 to 5). In each blending, the blended layouts are generated with $\alpha = 0.25, 0.5, 0.75$. This procedure produces a group of layouts in a grid arrangement. For the layouts in Figure 9, rows 2 to 4, although in each row, the layouts are not generated in the same blending procedure, they still exhibit obvious geometry and structure transitions.

Our method also enables a free exploration of a local layout space. Given a set of layouts, we first compute their pairwise distances. By performing Multi-Dimensional Scaling (MDS) [Wills et al. 2009], we will get a set of 2D points, each of which corresponds to an input layout. After a Delaunay triangulation [Barber et al. 1996] with these 2D points, we may explore this local layout space within the generated triangles. The layout at each position can be generated by a multiple layouts blending described in Section 3.4 (see Figure 1).

The correspondence estimation with our method is affected by the semantic information of the input layouts (see Rule 1). Given a pair of layouts, if their semantic labels are modified, the correspondence and the blended layouts generated by our method will also change. Figure 10 shows such an example, in which our method generates reasonable blended layouts with different inputs of semantic labels.

Fig. 10. Given a pair of layouts, our method can generate reasonable blended layouts with different inputs of semantic labels. The colors of the elements represent the semantic labels. The first two columns show the correspondence between the same pair of layouts with different semantic labels. The last three columns show the blended layouts generated by our method.

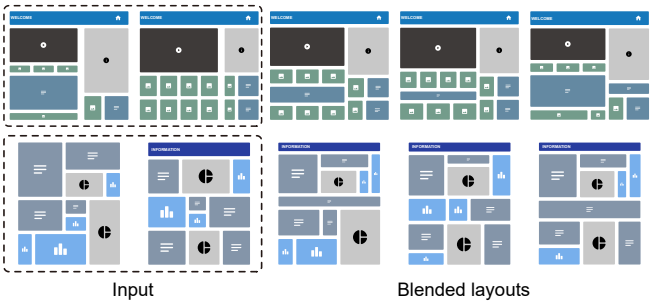

Fig. 11. Our method can generate plausible blended layouts with different types of input layouts, e.g., webpage layout and data report layout.

In the extreme case shown in Row 3 of Figure 10, only one pair of elements correspond to each other. Our method can still generate appropriate blended layouts, whose structures are combinations of the substructures of the input layouts. These results confirm the robustness of our method.

5 APPLICATIONS

Our method has many potential applications. Its capacity of creating novel layouts can be used for exploratory design of layouts (Section 5.1). The introduced layout similarity metric regularizes the layouts in a hyperspace. This helps jointly embed layouts and their corresponding line drawings in a latent space, which can be used for sketch-based layout retrieval (Section 5.2).

5.1 Exploratory design of layouts

Figure 12 shows our interface for exploratory design of novel layouts. The user may load layouts for blending. This interface was implemented in C++ with Qt UI library. In this prototype, we allow three layouts to be loaded since two layouts will produce blended layouts with limited variations and excessive layouts may confuse the user during the exploration. The bottom left window shows a navigation triangle whose vertices represent three input layouts. The black dot is the navigation point whose position determines the blending factors. The top left window shows one of the loaded layouts, which can be switched for display by selecting the corresponding vertices of the navigation triangle. The right window shows a blended layout. When the position of the navigation point changes, this blended layout will update in real time. Please refer to the accompanying video for layout blending in action.

To increase the diversity of the blended layouts, we add two local control tools. The first one is a local lock tool. During the navigation of blended layouts, the user may select a local arrangement of the current blended layout and issue a local lock command. The selected arrangement will then become fixed in the subsequent layout navigation. The second tool is a local navigation tool. Similar to the local lock tool, it allows the user to select a local arrangement and issue a local navigation command. Then a local navigation triangle with a local navigation point will appear adjacent to the selected arrangement. The user may modify the local blending factors by changing the location of the local navigation point.

We conducted a user study to verify the effectiveness of this prototype for exploratory design of novel layouts. The purpose was to investigate whether this interface helped users in designing novel layouts, and whether they got inspirations for layout design with this interface. In addition, we wanted to evaluate the plausibility of the blended layouts generated by our method.

Participants. We recruited 12 university students to participate in this study: 8 males and 4 females, aged 22 to 24. All of them had experiences in designing 2D layouts, e.g., creating slides or designing posters, etc. None of them was an expert designer. They got paid after the study as compensation.

Task. The task was free creation of novel layouts with selected ones. We filtered 675 high-quality layouts from the magazine layouts dataset [Zheng et al. 2019] as candidate layouts. We did not resize these layouts since they had similar shapes. The semantic information was considered to help the participants better evaluate the plausibility of the blended layouts. The participants were asked to freely select three layouts from the candidates and browse blended layouts. They were allowed to use the local control tools to generate more diverse blended layouts. During the browsing, they needed to select 5 desirable blended layouts. Each participant was asked to perform the above procedure 5 times with 5 triplets of layouts. In total there were 5 (selections) × 5 (triplets) × 12 (participants) = 300 (layouts) produced (see Figure 12. More layouts can be found in the supplemental materials).

Before the study, the participants were introduced to our interface to get familiar with the functions, including the global navigation and the local control tools. We explained to the participants about

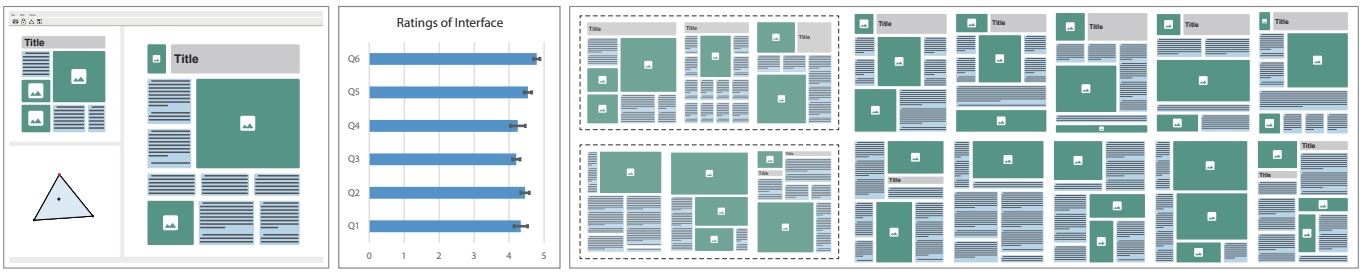

Fig. 12. Left: our interface for exploratory design of novel layouts. Middle: the average ratings of our interface from the user study participants. Each error bar represents the standard error of the mean. Right: two sets of layouts produced by the participants. The dashed frames highlight two groups of input layouts for blending, followed by the blended layouts selected by the participants.

the purpose of this interface, i.e., browsing blended layouts for inspiration, instead of creating a target layout in mind. We emphasized this with the participants until they claimed that they understood.

After the study, the participants filled out an questionnaire on their experiences of using this interface. The following questions were included: *Q1. Does this interface produce inspiring layouts? Q2. Is the way for browsing the layouts convenient? Q3. Are the local control tools useful for increasing the diversity of the produced layouts? Q4. Is the transition of the produced layouts during browsing smooth and continuous? Q5. Are the produced layouts reasonable blending of the selected ones? Q6. Do the blended layouts resemble real magazine layouts?* The participants needed to answer these questions in a 5-point Likert scale, with 1 being strongly negative and 5 being strongly positive.

Results. Figure 12 plots the statistics of the ratings of our interface according to the questionnaire. *Q1*, *Q2*, and *Q3* were about the functionality of the interface. Most participants were positive on the overall effectiveness (*Q1*, 4.3 on average, 83% above 3) and usability (*Q2*, 4.5 on average, 100% above 3). They also appreciated the design of the local control tools (*Q3*, 4.2, 100% above 3). *Q4*, *Q5*, and *Q6* were about the subjective rating of our layout blending method. Most participants were satisfied with the blended layouts produced by our method, including the blending stability (*Q4*, 4.3 on average, 83% above 3) and blending plausibility (*Q5*, 4.5 on average, 100% above 3; *Q6*, 4.8 on average, 100% above 3).

We also collected the feedback from the participants. Since they were not experts, their comments would better represent ordinary users, who would be more likely to be benefited from our technique. In general, the participants were positive about our interface. They commented that "*it is interesting to explore novel layouts by interpolating given ones*"; "*the transition between the blended layouts is smooth*"; "*the generated layouts are quite inspiring*"; etc. For improving the interface, they gave the following suggestions. First, some participants suggested including direct editing tools in the interface. They commented that "*I want to drag some elements to other locations, or remove them*"; "*It would be great if I can edit the sizes of the elements*". Second, several participants had alternative opinions on the rationality of some blended layouts. They commented that "*it looks strange if a title element appears at the bottom of the layout or has a small size*"; "*some figure elements have uncommon aspect ratios*". Third, a participant suggested further improving the local editing tools. He said that "*it would be great if I can select multiple branches at the same time*". Fourth, a few participants were not always satisfied with the blended layouts. They commented that "*I cannot always reach my target layout by browsing*"; "*there usually exists a gap between the generated layouts and my desired layout*". These comments reminded us to combine our interface with direct layout editing tools [Xu et al. 2019] for more convenient use, and incorporate semantic constraints [Bao et al. 2013] in the layout optimization to generate more realistic layouts. Interestingly, although we emphasized that the purpose of our interface with the participants before the study, i.e., our interface was designed for browsing blended layouts for inspiration instead of creating a target layout in mind, they still gave suggestions on creating specific layouts. This feedback implied that our interface was indeed effective at inspiring users in layout creation.

5.2 Sketch-based layout retrieval

Several works [Bunian et al. 2021; Huang et al. 2019] have already explored the task of layout retrieval, which provides intuitive tools for finding desired layouts in a large layout repository. The *Swire* system proposed by Huang et al. [2019] adopts sketches as the querying modality. It converts layouts into images for constructing a layout space. This conversion discards the important structural information of layouts, making the retrieval insensitive to structural similarity. Our method introduces a structural similarity measure, which helps construct a layout space with an emphasis on structures. If we embed the sketch representations of the layouts into this space, it will enable sketch-based layout retrieval. We demonstrate this idea to verify the feasibility of layout space construction with our structural similarity measure, and compare with *Swire* to show that our constructed space is structure-aware.

Networks. We reimplement the network of *Swire* for comparison. This network has two sub-networks, i.e., sketch VGG-A net and screenshot VGG-A net, and takes sketch-screenshot pairs as input for training. The loss for this network is a Triplet loss. To prepare the paired data for this network, we render each layout with two styles, i.e., sketch and textured image. The sketch is rendered by drawing the bounding boxes of the elements. The colors of the bounding boxes encode the semantic labels of the elements. The texture images are similar to those in Figures 1 and 12. Different elements are

distinguishable with this rendering. The training procedure follows its original setting with a training/testing ratio of 9:1.

To remove the impact of different network architectures and better examine the effect of our structural similarity measure, we borrow the network architecture of *Swire* to construct our layout space. Specifically, we only use the sketch VGG-A net since we only need to deal with the sketch input. The sketch set is the same as the one fed to *Swire*, with the same training/testing split. During the training, the positive and negative samples are determined based on our similarity measure, i.e., for each anchor sketch s, a positive sketch s^+ is selected from its top-50 nearest neighbors determined by our similarity measure and a negative sketch s^- is randomly selected from its top-1000 farthest neighbors. The Triplet loss is modified from *Swire*, formulated as:

$$L = D(s, s^+) + \max(0, m - D(s, s^-)), \tag{13}$$

with

$$\begin{aligned} D(s, s^*) &= \left\| f(s) - f(s^*) \right\|_2, \\ * &\in \{+, -\}, \end{aligned} \tag{14}$$

where $f(s)$ is the embedding of sketch s returned by the sketch VGG-A net. $m = 0.2$ is the margin between the positive and negative pairs and its value is same as that in *Swire*. The other settings of the training procedure, e.g., the optimizer and the learning rate, are also same as those in *Swire*.

Dataset. We adopt the magazine layouts dataset [Zheng et al. 2019] for the layout space construction. We first filter and process the layouts in this dataset to obtain a set of layouts (1,037 in total) with regular arrangements (i.e., well-defined structures). This dataset is then augmented to 10,370 layouts with our layout blending method.

Comparison. The layout retrieval with both *Swire* and our prototype is achieved by first passing a sketch to the sketch VGG-A net to obtain its latent code and then performing retrieval using a nearest neighbor search in the latent space. Since *Swire* and our prototype adopt the same network architecture, the evaluation on the neural network is not necessary. The main difference between *Swire* and our prototype is the similarity measure between layouts. *Swire* considers the layouts as textured images while our prototype emphasizes the structure of the layouts, and thus our prototype should be more structure-aware. However, there does not exist a ground truth data for this evaluation. Since the structure similarity is usually based on the human judgement [Patil et al. 2021], we conduct a simple perceptual study for the evaluation.

We randomly selected a set of sketches ($M = 10$) of the layouts from the testing set, fed them to *Swire* and our prototype, and collected the top-K retrieved layouts ($K = 5$) for each network. Each sketch and 10 retrieved layouts with this sketch formed a group and there were 10 groups in total. We created an online questionnaire to let people judge whether the input sketch and retrieved layouts had similar structures. In this questionnaire, each time a query sketch and a retrieved layout were displayed and people only needed to answer similar or not according to their structures. The order of the questions was randomized. 40 people ($N = 40$, 30 males and 10 females, aged 21 to 37) participated this study. We then computed the precision score for *Swire* and our prototype with the following equation respectively:

$$P = \frac{1}{MNK} \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{k=1}^{K} \delta_i(s_j, r_{jk}), \tag{15}$$

where s_j is a query sketch. r_{jk} is a retrieved layout with sketch s_j. $\delta_i(s_j, r_{jk})$ is the indicator function of a participant i. Its value is 0 or 1 based on the participant's judgement on the structural similarity between the sketch s_j and the retrieved layout r_{jk}.

The precision scores for *Swire* and our prototype were 32.0% and 63.2%, respectively. This result confirmed the feasibility of our structural similarity measure for constructing a structure-aware layout space. Figure 13 shows the retrieval results by feeding hand-drawn sketches to *Swire* and our prototype respectively. Although these results show the advantage of our prototype in structure-aware layout retrieval compared with *Swire*, they are not always satisfactory. This is mainly due to the concise nature of the sketch representation, especially when applying this representation to the retrieval of layouts with hierarchical structures. Further improving the sketch-based layout retrieval with more advanced learning techniques (e.g., graph neural network [Manandhar et al. 2020; Patil et al. 2021]) would be an interesting future direction.

6 CONCLUSION

In this paper, we have presented a novel method for blending layouts with hierarchical structures represented as layout trees. The core of our method is an optimal structure correspondence algorithm, which recursively finds optimal substructure correspondences. With this structurally consistent correspondence, we can unite the structures of the input layouts to construct a compound tree. Then a blended layout is obtained by extracting its intermediate structure from the compound tree and performing a layout optimization procedure. Our method also defines a structure-aware similarity measure between layouts. Our method has many potential applications. It provides a new way to create novel layouts from a small set of layouts. The structure-aware similarity measure can help construct a layout space, which may be integrated with learning-based methods [Arroyo et al. 2021; Kikuchi et al. 2021; Li et al. 2019b; Patil et al. 2020] for layout creation. We have demonstrated two applications in our paper, i.e., exploratory design of novel layouts and sketch-based layout retrieval. The effectiveness of the first application has been confirmed by the user study participants. The feasibility of the second application has been verified by our prototype.

Limitations. Our method still has room for improvement. Our structural correspondence may not be always preferable. Figure 14 shows an example (top) in which the correspondence may not be agreed by all people. This is because two visually similar layouts may have distinct hierarchical structures, which are not always notable to some people. Our similarity measure heavily relies on the hierarchical structures. If two same layouts have different hierarchical structures (Figure 14, bottom), their similarity error would be large. This is quite controversial since no evidence shows which aspect, visual appearance or structure, is more significant in defining the similarity between layouts. It is possible to combine these two kinds of measures to define a more comprehensive one. Another limitation of our method is that we only consider relations between sibling

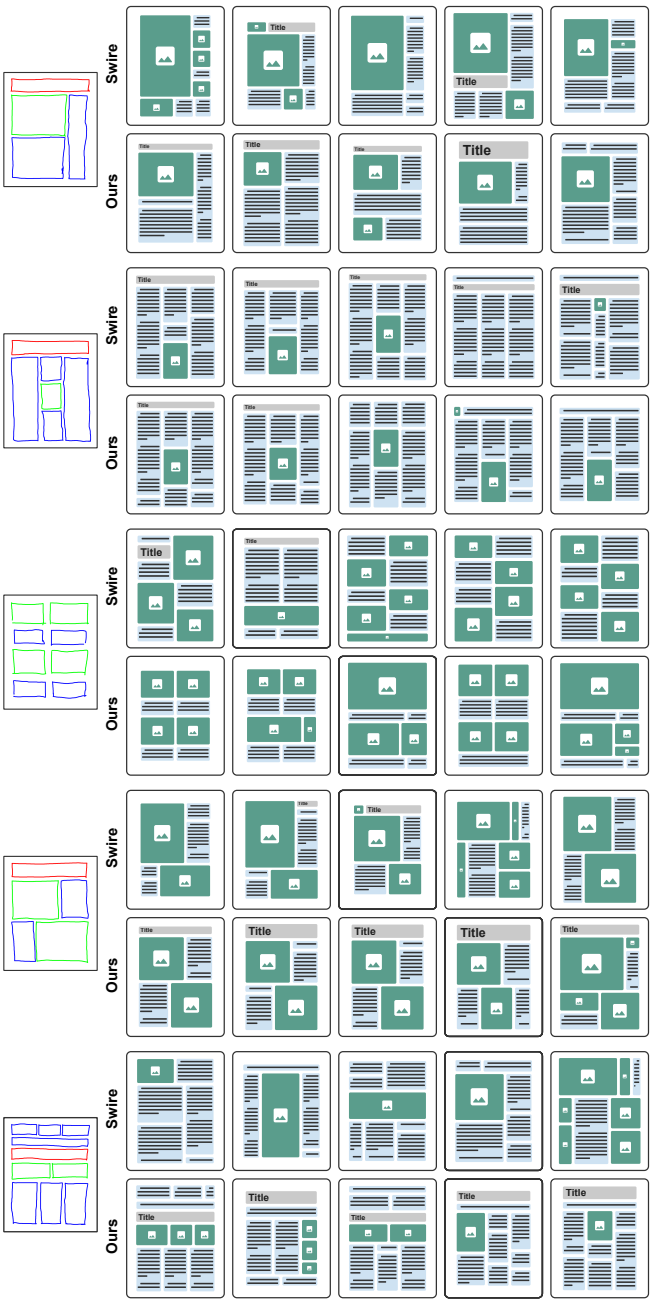

Fig. 13. By feeding a hand-drawn sketch (Column 1), *Swire* and our prototype can return layouts (Columns 2 to 6) as retrieval results. These results show the advantage of our prototype in structure-aware layout retrieval compared with *Swire*.

vertices in a layout tree. This may lead to misalignment between elements in different substructures (e.g., the layout in Row 4, Column 1 of Figure 9). We do not consider relations between elements in different substructures since they may lead to conflict of constraints. Our current implementation only considers alignment, equal-spacing,

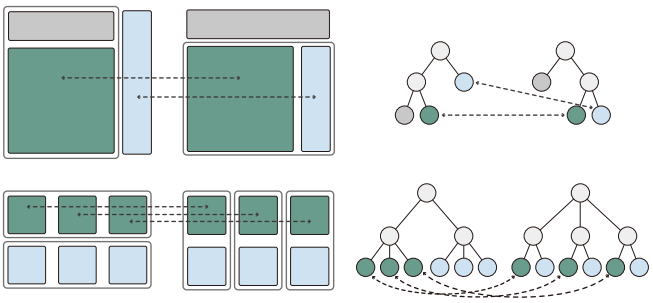

Fig. 14. Two examples that our method produces controversial correspondences. Top: A correspondence that may not be agreed by all the people. Bottom: A correspondence between two similar layouts with distinct hierarchical structures. The colors of the elements represent the semantic labels.

and equal-size relations. More advanced relations would further improve our method, e.g., symmetry relation would help generate more regular blended layouts; containing relation would enable our method to process layouts with overlapped elements.

Future work. Our method may lead to several future directions. First, our method for estimating hierarchical structure correspondence may be used to solve other correspondence problems, such as partial matching in shape correspondence. It is also possible to extend our recursive substructure correspondence algorithm for retrieving objects which are under hierarchical organizations. Another potential future work is to incorporate our hierarchical structure similarity measure with learning techniques [Arroyo et al. 2021; Kikuchi et al. 2021; Li et al. 2019b; Patil et al. 2020] for automatic or sketch-based layout creation. It would also be promising to further extend our method to estimate the correspondence between layouts that have flexible hierarchical structures. Our method does not consider the detailed contextual information, e.g., the specific texts and figures, for the layout generation. It would be promising to consider such information in the future. At last, we only invited non-expert users to test our framework of exploratory design of novel layouts. It would be interesting to invite professional designers to try our framework for further improvement in the view of both interface design and layout creation procedure.

ACKNOWLEDGMENTS

We thank the reviewers for their constructive comments, the user study participants for their time. This project was supported by NSFC (62072316, 62161146005, U2001206), Shenzhen Science and Technology Program (20200812104508001, RCJC20200714114435012, JCYJ20210324120213036), the Centre for Applied Computing and Interactive Media (ACIM) of School of Creative Media, City University of Hong Kong, and Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ).

REFERENCES

Ibraheem Alhashim, Honghua Li, Kai Xu, Junjie Cao, Rui Ma, and Hao Zhang. 2014. Topology-varying 3D shape creation via structural blending. *ACM Trans. Graph.* 33, 4 (2014), 1–10.

Ibraheem Alhashim, Kai Xu, Yixin Zhuang, Junjie Cao, Patricio Simari, and Hao Zhang. 2015. Deformation-driven topology-varying 3D shape correspondence. *ACM Trans.*

Graph. 34, 6 (2015), 1–13.
Rahul Arora, Ishan Darolia, Vinay P Namboodiri, Karan Singh, and Adrien Bousseau. 2017. Sketchsoup: Exploratory ideation using design sketches. *Comput. Graph. Forum* 36, 8 (2017), 302–312.
Diego Martin Arroyo, Janis Postels, and Federico Tombari. 2021. Variational Transformer Networks for Layout Generation. In *CVPR*. 13642–13652.
Greg J Badros, Alan Borning, and Peter J Stuckey. 2001. The Cassowary linear arithmetic constraint solving algorithm. *ACM Trans. Comput.-Hum. Interact.* 8, 4 (2001), 267–306.
Fan Bao, Dong-Ming Yan, Niloy J Mitra, and Peter Wonka. 2013. Generating and exploring good building layouts. *ACM Trans. Graph.* 32, 4 (2013), 1–10.
C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. 1996. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.* 22, 4 (1996), 469–483.
Patrick Baudisch, Edward Cutrell, Ken Hinckley, and Adam Eversole. 2005. Snap-and-go: helping users align objects without the modality of traditional snapping. In *CHI*. 301–310.
Eric A Bier and Maureen C Stone. 1986. Snap-dragging. *SIGGRAPH Comput. Graph.* 20, 4 (1986), 233–240.
Alan Borning, Kim Marriott, Peter Stuckey, and Yi Xiao. 1997. Solving linear arithmetic constraints for user interface applications. In *UIST*. 87–96.
Sara Bunian, Kai Li, Chaima Jemmali, Casper Harteveld, Yun Fu, and Magy Seif Seif El-Nasr. 2021. VINS: Visual Search for Mobile User Interface Design. In *CHI*. 1–14.
Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *UIST*. 845–854.
Michael Goesele, Jens Ackermann, Simon Fuhrmann, Carsten Haubold, Ronny Klowsky, Drew Steedly, and Richard Szeliski. 2010. Ambient point clouds for view interpolation. *ACM Trans. Graph.* 29, 4 (2010), 1–6.
Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.
Kamal Gupta, Justin Lazarow, Alessandro Achille, Larry S Davis, Vijay Mahadevan, and Abhinav Shrivastava. 2021. Layouttransformer: Layout generation and completion with self-attention. In *ICCV*. 1004–1014.
Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. 2020. Graph2plan: Learning floorplan generation from layout graphs. *ACM Trans. Graph.* 39, 4 (2020), 118–1.
Forrest Huang, John F Canny, and Jeffrey Nichols. 2019. Swire: Sketch-based user interface retrieval. In *CHI*. 1–10.
Qi-Xing Huang, Bart Adams, Martin Wicke, and Leonidas J Guibas. 2008. Non-rigid registration under isometric deformations. *Comput. Graph. Forum* 27, 5 (2008), 1449–1457.
Arjun Jain, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel. 2012. Exploring shape variations by 3d-model decomposition and part-based recombination. *Comput. Graph. Forum* 31, 2pt3 (2012), 631–640.
Yue Jiang, Ruofei Du, Christof Lutteroth, and Wolfgang Stuerzlinger. 2019. ORC layout: Adaptive GUI layout with OR-constraints. In *CHI*. 1–12.
Yue Jiang, Wolfgang Stuerzlinger, and Christof Lutteroth. 2021. ReverseORC: Reverse Engineering of Resizable User Interface Layouts with OR-Constraints. In *CHI*. 1–18.
Yue Jiang, Wolfgang Stuerzlinger, Matthias Zwicker, and Christof Lutteroth. 2020. ORCSolver: An Efficient Solver for Adaptive GUI Layout with OR-Constraints. In *CHI*. 1–14.
Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. 2019. Layoutvae: Stochastic scene layout generation from a label set. In *ICCV*. 9895–9904.
Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph.* 31, 4 (2012), 1–11.
Youwen Kang, Zhida Sun, Sitong Wang, Zeyu Huang, Ziming Wu, and Xiaojuan Ma. 2021. MetaMap: Supporting Visual Metaphor Ideation through Multi-dimensional Example-based Exploration. In *CHI*. 1–15.
Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. 2021. Constrained graphic layout generation via latent optimization. In *ACM International Conference on Multimedia*. 88–96.
Yanir Kleiman, Oliver van Kaick, Olga Sorkine-Hornung, and Daniel Cohen-Or. 2015. SHED: shape edit distance for fine-grained shape similarity. *ACM Trans. Graph.* 34, 6 (2015), 1–11.
Janin Koch and Antti Oulasvirta. 2016. Computational layout perception using gestalt laws. In *CHI Extended Abstracts*. 1423–1429.
Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2, 1-2 (1955), 83–97.
Ranjitha Kumar, Jerry O Talton, Salman Ahmad, and Scott R Klemmer. 2011. Bricolage: example-based retargeting for web design. In *CHI*. 2197–2206.
Hsin-Ying Lee, Lu Jiang, Irfan Essa, Phuong B Le, Haifeng Gong, Ming-Hsuan Yang, and Weilong Yang. 2020. Neural design network: Graphic layout generation with constraints. In *ECCV*. Springer, 491–506.
Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. 2019b. LayoutGAN: Generating Graphic Layouts with Wireframe Discriminators. In *International Conference on Learning Representations*.
Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, Daniel Cohen-Or, and Hao Zhang. 2019a. Grains: Generative recursive autoencoders for indoor scenes. *ACM Trans. Graph.* 38, 2 (2019), 1–16.
Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. 2015. SMPL: A skinned multi-person linear model. *ACM Trans. Graph.* 34, 6 (2015), 1–16.
Dipu Manandhar, Dan Ruta, and John Collomosse. 2020. Learning structural similarity of user interface layouts using graph networks. In *ECCV*. Springer, 730–746.
Akshay Gadi Patil, Omri Ben-Eliezer, Or Perel, and Hadar Averbuch-Elor. 2020. Read: Recursive autoencoders for document layout generation. In *CVPR*. 544–545.
Akshay Gadi Patil, Manyi Li, Matthew Fisher, Manolis Savva, and Hao Zhang. 2021. LayoutGMN: Neural Graph Matching for Structural Layout Similarity. In *CVPR*. 11048–11057.
Roope Raisamo and Kari-Jouko Räihä. 1996. A new direct manipulation technique for aligning objects in drawing programs. In *UIST*. 157–164.
William T Reeves. 1981. Inbetweening for computer animation utilizing moving point constraints. *SIGGRAPH Comput. Graph.* 15, 3 (1981), 263–269.
Alec Rivers, Takeo Igarashi, and Frédo Durand. 2010. 2.5 D cartoon models. *ACM Trans. Graph.* 29, 4 (2010), 1–7.
Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.
Stefan Schwarz, Mateusz Pawlik, and Nikolaus Augsten. 2017. A new perspective on the tree edit distance. In *International Conference on Similarity Search and Applications*. Springer, 156–170.
Wolfgang Stuerzlinger, Olivier Chapuis, Dusty Phillips, and Nicolas Roussel. 2006. User interface façades: towards fully adaptable user interfaces. In *UIST*. 309–318.
Sou Tabata, Hiroki Yoshihara, Haruka Maeda, and Kei Yokoyama. 2019. Automatic layout generation for graphical design magazines. In *ACM SIGGRAPH 2019 Posters*. 1–2.
Daniel Vlasic, Matthew Brand, Hanspeter Pfister, and Jovan Popović. 2005. Face Transfer with Multilinear Models. *ACM Trans. Graph.* 24, 3 (jul 2005), 426–433.
Josh Wills, Sameer Agarwal, David Kriegman, and Serge Belongie. 2009. Toward a perceptual space for gloss. *ACM Trans. Graph.* 28, 4 (2009), 1–15.
Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. 2019. Data-driven interior plan generation for residential buildings. *ACM Trans. Graph.* 38, 6 (2019), 1–12.
Pengfei Xu, Hongbo Fu, Chiew-Lan Tai, and Takeo Igarashi. 2015. Gaca: Group-aware command-based arrangement of graphic elements. In *CHI*. 2787–2795.
Pengfei Xu, Guohang Yan, Hongbo Fu, Takeo Igarashi, Chiew-Lan Tai, and Hui Huang. 2019. Global Beautification of 2D and 3D Layouts with Interactive Ambiguity Resolution. *IEEE TVCG* (2019).
Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.* 18, 6 (1989), 1245–1262.
Xinru Zheng, Xiaotian Qiao, Ying Cao, and Rynson WH Lau. 2019. Content-aware generative modeling of graphic design layouts. *ACM Trans. Graph.* 38, 4 (2019), 1–15.
Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. 2019. Publaynet: largest dataset ever for document layout analysis. In *International Conference on Document Analysis and Recognition*. IEEE, 1015–1022.
Chenyang Zhu, Renjiao Yi, Wallace Lira, Ibraheem Alhashim, Kai Xu, and Hao Zhang. 2017. Deformation-driven shape correspondence via shape recognition. *ACM Trans. Graph.* 36, 4 (2017), 1–12.