Global Beautification of 2D and 3D Layouts with Interactive Ambiguity Resolution

Pengfei Xu, Guohang Yan, Hongbo Fu, Takeo Igarashi, Chiew-Lan Tai, and Hui Huang

Abstract—Specifying precise relationships among graphic elements is often a time-consuming process with traditional alignment tools. Automatic beautification of roughly designed layouts can provide a more efficient solution but often lead to undesired results due to ambiguity problems. To facilitate ambiguity resolution in layout beautification, we present a novel user interface for visualizing and editing inferred relationships through an automatic global layout beautification process. First, our interface provides a preview of the beautified layout with inferred constraints without directly modifying an input layout. In this way, the user can easily keep refining beautification results by interactively repositioning and/or resizing elements in the input layout. Second, we present a gestural interface for editing automatically inferred constraints by directly interacting with the visualized constraints via simple gestures. Our technique is applicable to both 2D and 3D global layout beautification, supported by efficient system implementation that provides instant user feedback. Our user study validates that our tool is capable of creating, editing and refining layouts of graphic elements, and is significantly faster than the standard snap-dragging or command-based alignment tools for both 2D and 3D layout tasks.

Index Terms—Global beautification, layout editing, snapping, alignment, ambiguity resolution, gestural interface.

1 Introduction

Specifying precise relationships, such as alignment and equal-spacing between graphic elements, might be one of the most fundamental operations in creating 2D or 3D layouts. 2D layouts are commonly achieved in commercial software packages, like Adobe Illustrator and Microsoft PowerPoint, by using command-based arrangement tools (e.g., issuing a command to equally space the selected elements horizontally) and/or direct positioning aided by snapping. To arrange 3D shapes in a scene, commercial software packages, such as Autodesk 3ds Max, usually adopt a multiple viewport interface to transform a 3D arrangement into 2D arrangements in three projection planes (see the accompanying video). In each projection plane, 2D arrangement commands and snapping tools are usually employed.

Snapping might be the simplest beautification technique. It first infers spatial relationships between an element being manipulated and each of the existing elements, and then provides snapping suggestions to interactively achieve desired relationships [1]. We classify snapping as a *local beautification* technique since each time only a current element is being beautified while all existing elements are kept fixed, leading to an *element-by-element* beautification process. Due to its local nature, snapping itself is not very effective for designing constrained global patterns (e.g., equal-spacing patterns with the ending elements aligned, as shown in Figure 1). Therefore, snapping-based alignment tools are often used together with command-based tools. As we will discuss shortly, such traditional tools are rather tedious and require a carefully ordered set of manual operations to achieve a desired layout.

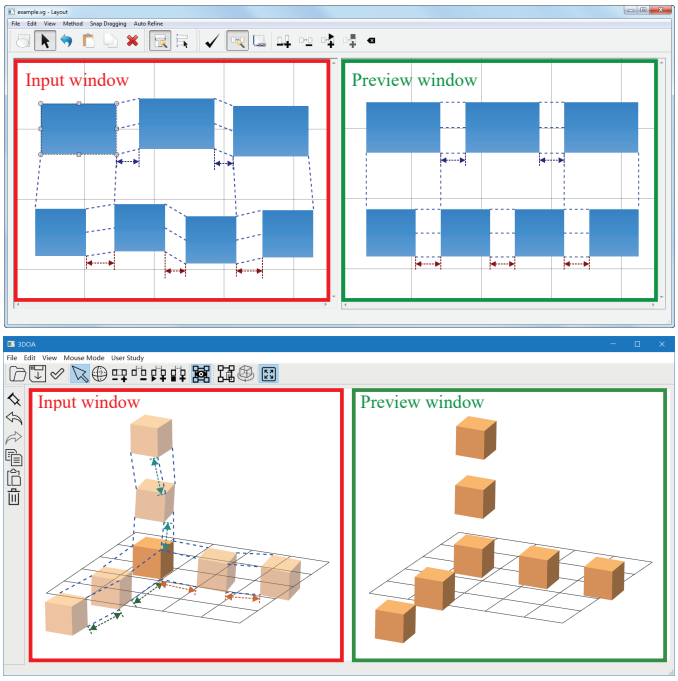


Fig. 1. Our novel interface for global beautification of layouts (2D top, 3D bottom) with the power of interactive ambiguity resolution. Grids are visualized only for view reference purpose (i.e., not for editing).

On the other hand, humans are able to unambiguously tell the desired layouts of graphic elements by viewing all elements as a whole. This motivates us to design a tool for *global beautification* of layouts of graphic elements, i.e., to first infer perceptually meaningful relationships among a set of roughly placed graphic elements and then refine

- *P. Xu (xupengfei.cg@gmail.com), G. Yan (guohang.yan@gmail.com), and H. Huang (hhzhiyan@gmail.com) are with the Visual Computing Research Center (VCC), and Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen University.*
- *H. Fu (hongbofu@cityu.edu.hk) is with the City University of Hong Kong.*
- *T. Igarashi (takeo@acm.org) is with the University of Tokyo.*
- *C. Tai (taicl@cs.ust.hk) is with the Hong Kong University of Science & Technology.*
- *H. Huang is the corresponding author of this paper.*

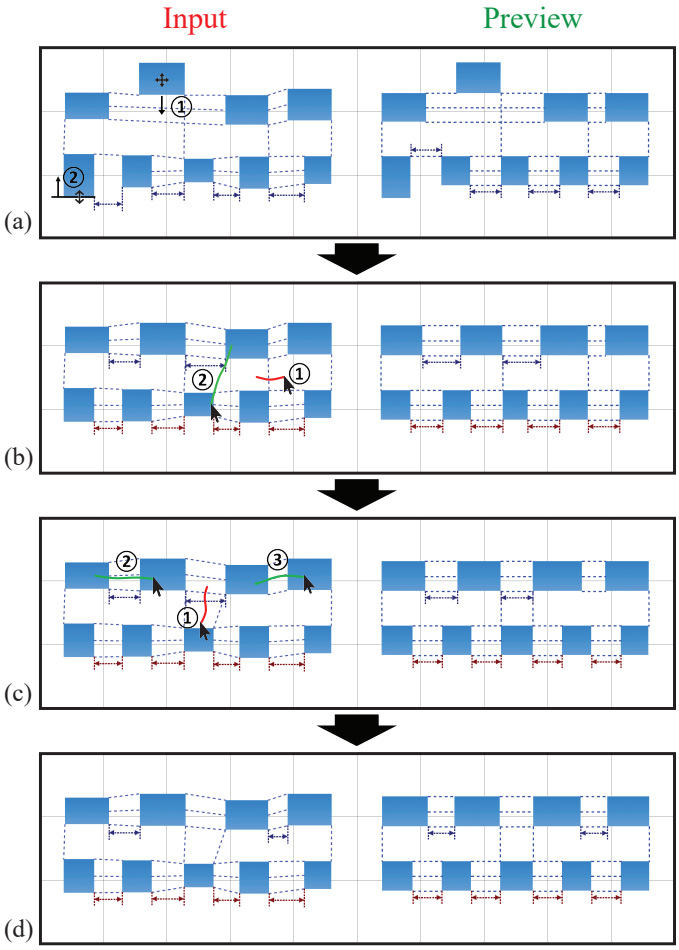


Fig. 2. To refine a beautified layout, as previewed in right, the user may directly reposition (a-1) or resize (a-2) individual elements, or remove inferred (b-1) or add new (b-2) edge-alignment relationships, or remove inferred (c-1) or add new (c-2 and c-3) equal-spacing relationships.

their positions and sizes to get a well-aligned layout that involves only small changes to the input layout. While similar concepts of global beautification have been proposed for sketch-based user interfaces, the existing methods (e.g., [2], [3]) simply apply global beautification results directly to elements being edited, as often done for local beautification interfaces. Early beautification of elements being edited would prevent the user from placing them freely to form a global pattern, thus disturbing the process of layout design. In addition, the patterns formed by the elements may involve ambiguities. The widely-adopted suggestion-based interfaces [4], [5], [6] are not directly applicable here, since a handful of ambiguous relationships may easily lead to an excessive number of suggestions.

We present a novel user interface for addressing a commonly known ambiguity problem in global layout beautification. As shown in Figure 1, our interface shows a preview of the beautification, without immediately modifying any input element. In our prototype, this preview is displayed in a separated window. While this interface is simple, it has the following benefits. First, the user can focus on the layout design by manipulating individual elements in the input window. Second, the user might refine the beautification results by slightly modifying the input elements. To facilitate a more direct control of the beautification results, we also present a gestural interface, which allows easy editing of automatically inferred constraints, i.e., desired relationships between elements in the input window. See Figure 2 and the accompanying video for live demos.

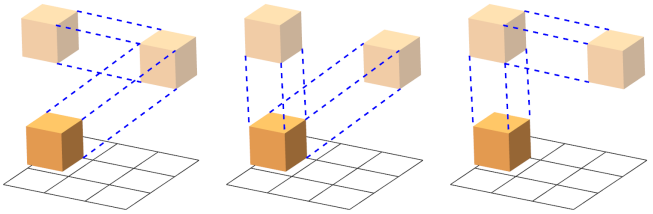


Fig. 3. The elements in three layouts have the same projected positions. The visualized relations (dashed lines) help the perception of their relative positions in 3D.

Comparing with 2D layout creation, arranging 3D elements could be much more tedious since the additional dimension will introduce extra arrangement operations. Besides, more ambiguities will appear during the interaction since 3D elements are displayed as projected 2D shapes, losing the certainty of depth. These problems can be greatly alleviated by extending our proposed user interface to 3D elements arrangement. By detecting and visualizing the relations between the projected elements in the screen space, the perception ambiguities in the arranged elements can be eliminated (see Figure 3). In addition, enforcing the detected relations by optimization will ease the tedious elements manipulation procedure since placing the projected elements in screen space to achieve 2D patterns requires much fewer operations than 3D elements manipulation.

We present an effective development of the proposed global layout beautification interface for both 2D and 3D layout tasks. Our tool allows interactive refinement of graphic elements and/or constraints with instant beautification feedback. We have conducted three studies to verify the effectiveness of our interface. First, we ran a study to evaluate the effectiveness of beautification preview. Second, we ran a study to evaluate whether our tool could replace the standard snapping and alignment commands in 2D elements arrangement tasks, omitting other layout helper features (e.g., equal-sizing, auto-align) which can peacefully coexist with our tool. Third, we ran a study to evaluate whether our tool is more user-friendly compared to a traditional multi-viewpoint interface with snapping in 3D elements arrangement tasks. The results show that our tool provided an efficient way for creating, editing, and refining the 2D test layouts and a more natural way for arranging 3D elements. The intuitiveness and ease-of-use of our interface were also confirmed by user participants.

2 Related Work

Traditional alignment techniques. It is hard to track down the history of shape alignment tools. Command-based alignment tools might be one of the most common ways to align objects [7]. They adopt a two-step procedure: first select a group of objects to be aligned and then issue a certain arrangement command (e.g., left-align or equally space the selected objects). The group-aware arrangement commands introduced by Xu et al. [8] improve the traditional arrangement commands by allowing the alignment

of multiple groups of elements with one single selection. However, their technique is not able to optimize an entire layout in a global manner. Snapping is another widely used technique for aligning 2D [1], [9] or 3D elements [10], [11]. It provides aligned positions by snapping an object to either the background grids, manually created guides, or other objects [12]. Snapping supports direct manipulation and is thus more intuitive to use, while command-based tools are more effective in aligning multiple objects simultaneously and have a better control of global alignment. Given their unique advantages, these traditional tools collectively are capable of creating very complex well-aligned layouts of graphic elements. However, even a simple arrangement task as shown in Figure 1 already demands a series of operations with the traditional alignment tools. The repeated use of operations like snapping, element selection, and command selection is tedious and error-prone, especially for complex layouts. In addition, since different ordering of such operations might lead to different results, a user has to plan a series of operations beforehand and pay special attention to their order during editing. This somewhat diverts the user from the tasks of layout design and creation.

Automatic/semi-automatic layout techniques. There exist many layout helper features beyond the basic snapping and alignment commands, and some have already been integrated into commercial editors. For example, many algorithms have been proposed for automatic or semi-automatic graph layout generation [13], [14], [15], e.g., the *auto-align* and *auto-space* tools in Microsoft Visio. However, they often significantly change the layout of the input elements without attempting to infer and maintain the underlying patterns in the input layout. The equal-sizing feature in Microsoft Visio allows easy size-adjustment of selected elements. Several editors such as OmniGraffle and PowerPoint support easy creation of equally spaced elements when duplicating an element multiple times. These helper features are very efficient for only specific tasks and are orthogonal to our tool.

Beautification techniques. Our work is inspired by previous efforts on beautification that also aims to bring as small change to the initial input as possible. There are plenty of beautification techniques mainly developed for sketch-based user interfaces to tolerate errors from freehand input sketches. These existing techniques can be largely categorized into two groups [16]: *local beautification* and *global beautification*. Similar to snap-dragging [1], local beautification of freehand sketches beautifies a current input stroke based on either itself (often through sketch recognition [17], [18], [19], [20]) or its geometric relationship(s) to the existing elements (e.g., [4], [21], [22]). A typical user interface for local beautification achieves beautification progressively, stroke by stroke. Each stroke is replaced with its beautified version and kept fixed during the beautification of subsequent strokes. Due to the local nature, such interfaces are often easy to control. However, unfortunately, local beautification is not suitable for the beautification of layouts, which should be treated globally.

Compared to local beautification, global beautification has been insufficiently explored. Pavlidis and Van Wyk [3] present the first automatic beautifier for drawings and illustrations. Their algorithm focuses on inferring and enforcing relations among points and lines, and thus can be extended for our layout beautification problem by considering special requirements like how to preserve aspect ratios of elements. However, they do not give details about the user interface of their system. Bolz [23] presents for the first time a user interface for global beautification of drawings. It concentrates on how to deal with parameter settings (e.g., via a parameter menu). Due to possibly frequent parameter changes, the beautifier by default is manually activated each time, though it is discussed that automatic activation of the beautifier (like ours) is a desired feature. Bolz's system overlays the beautified drawing with an outline of the unmodified version for some seconds. A similar visualization of the original and beautified versions is adopted in [2], which however deliberately defers the beautifier's feedback. Otherwise, the user may easily get disturbed by beautification results. In contrast, our user interface allows instant feedback with minimal disturbance. Cheema et al. [24] present a novel sketch beautification algorithm but adopt a simple interface similar to Bolz's. Jiang et al. [25] solve a similar layout regularization problem arising from facade images. However, they focus more on an automatic 2D layout regularization system instead of a general interactive framework for layout beautification. In brief, none of the above methods has explored an interface for visualizing and editing constraints.

Techniques utilizing geometric features. Our optimization strategy for determining valid constraints bears some resemblance to the previous solutions for identifying geometric features towards sketch recognition. For example, Veselova and Davis [26] employ studies of human perception to determine which geometric features are the most important for symbol recognition. Hammond and Davis [27] present a graphical debugging tool for learning structural descriptions from automatically generated near-miss examples. Similar to earlier work [3], [4], we adopt an automatic, iterative approach to avoid dramatic changes to the input layout.

Techniques for ambiguity resolution. It is not always possible to infer desired relationships among roughly placed elements, especially when there are ambiguities. A possible solution for ambiguity resolution is to provide multiple alternatives as suggestions from which the user can then choose [4], [5], [6]. The effectiveness of such suggestion-based interfaces has been demonstrated for progressive beautification [4], [6]. It is unclear whether suggestions can be used to effectively resolve ambiguity from global beautification since a handful of ambiguous relationships may easily lead to a large set of suggestions. Instead of attempting to select the best suggestions from a big pool, we design a gestural interface for directly editing a limited number of constraints.

Constraint-based systems. Constraint-based systems have been extensively studied and a detailed review is beyond the scope of this paper. Most constraint-based systems require the user to specify constraints explicitly (e.g., [28], [29], [30]). Instead, our beautification interface focuses on automatic inference of geometric constraints, and thus only very few edits on constraints (after roughly placing elements) are needed to achieve a desired layout. Note that our current system performs once-off alignment only and does

not maintain inferred constraints during subsequent editing [31]. Recently Zeidler et al. [32] introduce a novel layout preview but focus on the resizing behavior of a constraint-based layout during the design phase. Since constraint-based systems often need to deal with various linear and nonlinear constraints, possibly with complex interactions among them, significant effort has been devoted to the topic of constraint solving (e.g., [33], [34], [35], [36]). Such solvers can be potentially used for our beautification optimization problem.

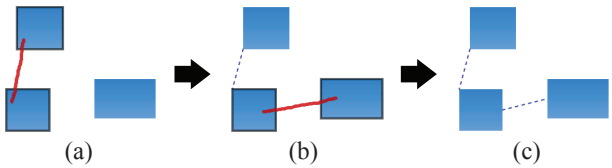


Fig. 4. Two examples of our gestural interface for adding alignment constraints. An appropriate constraint is automatically determined by examining the relative location of two involved elements and the position of the stroke's ending points relative to the elements.

3 User Interface

In this section, we first describe our main interface for global beautification and then present our gestural interface for editing constraints. To make it simple, we first assume that the beautifier is already activated and applied to all input elements, and will discuss how our beautifier can coexist with other tools at the end of this section.

3.1 Global Beautifier Interface

Our global beautifier takes a set of roughly placed elements as input and returns a well-aligned layout as output. Since the desired relationships might not be immediately clear during editing, even to a human viewer, we design a user interface that does not directly change the location and size of elements in the input layout but rather gives a preview of the refined layout in another window (Figure 1). In this way, the user can focus on layout design itself by placing elements roughly, without being disturbed by the beautifier's feedback.

Since the input layout is not immediately replaced with the beautified version, the user may keep editing the current layout by translating or resizing individual elements, and our beautifier gives instant feedback in the preview window. This simple interface is suitable for creating, editing or refining layouts. At any point, the user may confirm to apply the beautified layout to the input one.

3.2 Gestural Interface for Editing Constraints

Automatically inferring underlying relationships among elements plays a vital role in layout beautification. Since relationships are formulated as constraints in the beautification optimization, we will use the terms relationships and constraints interchangeably in the following discussion.

Automatically inferred relationships might not always be wholly desirable. This problem might be *indirectly* solved by the interactive refinement of certain elements' position and/or size. Alternatively, the user may *directly* add or remove relationships using a gestural interface as shown in Figure 2 (b) and (c). Our implemented beautifier currently supports two types of geometric relationships: edge alignment and equal-spacing.

At any time the user may edit constraints in the input window, where he or she may edit elements for interactive refinement. To facilitate the editing of constraints, we display all automatically inferred relationships using dashed lines and arrows in both the input and preview windows (Figure 1). The visualized constraints in the input window are editable via the gestural interface while those in the preview window are read-only for examining the currently achieved relationships. In real-world applications, constraints may be displayed on demand to avoid visual clutter, since in general only few constraint edits are needed.

The user enters the mode for editing edge alignment constraints by clicking an "alignment constraint editing" button, which might be replaced with a gesture in the future implementation. In this mode, repositioning and resizing of individual elements are disabled. To remove an existing alignment constraint, the user performs a cutting gesture on a constraint of interest (Figure 2 (b-1)), i.e., drawing a stroke roughly perpendicular to the dashed line corresponding to the constraint. To add a new alignment constraint, the user simply draws a stroke connecting a pair of elements of interest (Figure 2 (b-2)). Our interface automatically determines whether a horizontal or vertical alignment constraint is needed by examining the relative location of the two elements. For example, the stroke in Figure 4 (a) is interpreted as an intention for vertical alignment while the stroke in Figure 4 (b) is for horizontal alignment. For horizontal alignment, the user may indicate the preference for top-, middle- or bottom-align by placing the ending points of the stroke into the respective parts of the elements. For this reason, the stroke in Figure 4 (b) is recognized for achieving middle alignment. A similar control exists for vertical alignment (see an example in Figure 4 (a)). To make the interface more user-friendly, we may put three control handles (e.g., small translucent circles) on each side of an element, indicating that the user can drag from one handle to another to set alignments. The mode for editing alignment constraints is deactivated when the user switches to other editing modes.

For simplicity, the activation and deactivation of our gestural tool for editing equal-spacing constraints are achieved via mode-switching buttons. To remove an equal-spacing constraint the user performs a cutting gesture (Figure 2 (c-1)). To add a new equal-spacing constraint, the user invokes multiple (at least two) strokes, with each of them connecting a pair of elements (Figure 2 (c-2 and c-3)). Constraints for equal-spacing between the specified pairs of elements will then be added. Horizontal or vertical equal-spacing is automatically determined by looking into the relative position of pairs of elements.

Not all automatically inferred constraints can be enforced at the same time. Our interface only visualizes the constraints that can coexist. However, the user may introduce constraints that conflict with the existing ones. To capture the user's latest intention, any constraint that is in conflict with the newly added constraint will be removed. Thus in some cases, the user may see one or more existing

constraints removed due to the new constraints.

3.3 Interface for 3D Element Arrangement

Creating 3D layouts involves the alignment of elements in the additional depth dimension. A straightforward extension of our 2D beautification approach to 3D is to first ask users to roughly place the elements in the 3D space and then directly detect the 3D relations among these elements. Although this simple solution might reduce the number of operations for the precise 3D position and orientation specification, it still requires direct manipulation of elements in 3D, which is a tedious process, especially for novice users. With a 2D display device, we notice that the perception of 3D layouts is actually based on the rendered elements in the 2D screen space. It implies that 3D relations could be inferred from the projected 2D elements. This motivated us to design and develop a 2D interface for 3D layout creation and beautification.

Our interface allows a user to manipulate the projected elements in the screen space. By arranging the projected elements, the user can form a layout which can be perceived as a desired 3D layout (see Figure 3). The global beautifier takes the projected elements as input and infers the 2D relations among them. These 2D relations are then further interpreted as 3D relations among 3D elements by a back projection. These 3D relations are visualized to the user for assisting the perception. The global beautifier also returns a well-aligned 3D layout as output in the preview window (Figure 1 bottom). The user can change the view in the preview window to better examine the 3D layout. If the generated 3D layout is not the desired one, the user can modify the layout by rearranging the projected elements or using the gestural interface to edit the current relations.

Note that the relations among 3D elements can only determine their relative 3D positions. To determine the absolute 3D positions, the user can select some elements as *anchors*. These anchor elements possess exact 3D positions and can be manipulated in the 3D space with traditional tools. The other elements, which we call *floating elements*, have incomplete 3D information and can be manipulated in the screen space. The 3D positions of these floating elements are inferred when the user accepts the beautified 3D layout. The user can also change the view in the input window for better perception during the arrangement. The projected 2D position of the floating elements remains unchanged, but their orientations alter with the camera. The user can also change an anchor element to a floating element to remove its 3D position information (see the accompanying video for a clear illustration).

4 Implementation

Now we describe how we implement the proposed beautifier interface, which essentially follows the general beautification framework proposed by Pavlidis and Van Wyk [3]. The details are listed here for easy reproduction.

4.1 Pattern Detection

We focus only on edge alignment and equal-spacing relationships, since they already enable all the functions of snapping and alignment commands. We do not consider face alignment in 3D arrangement explicitly since it could be implicitly achieved by two edge alignment relationships. For illustration, we always use rectangles to represent 2D graphic elements and cuboids to represent 3D graphic elements since element alignment is often achieved at the bounding box level.

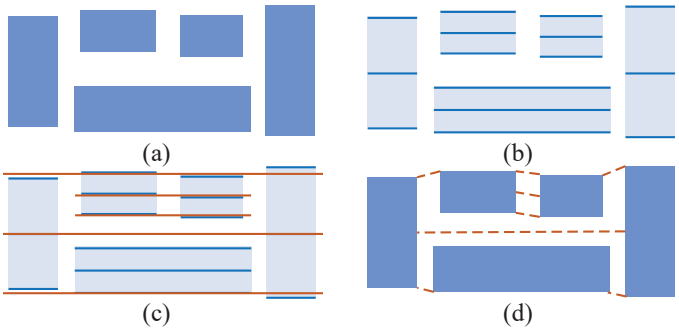


Fig. 5. Illustration for inferring relationships (dashed lines) for horizontal alignment of 2D elements.

Inferring edge-alignment relationships. The alignment relations are inferred by detecting collinearity relations among the 2D edges, which are extracted from the bounding boxes of 2D elements or the 2D projected bounding boxes of 3D elements. For 2D elements, we also include the horizontal and vertical middle edges (i.e., a horizontal or vertical segment passing through the center) of the bounding box, following the snapping tools in existing editors like Microsoft Visio. Given a set of extracted 2D edges, detecting collinearity relations is essentially a clustering problem, where a set of collinear lines form a cluster. Although there exist many general techniques for such clustering problems, most of them require indicating the scale of the cluster by parameters, which in most techniques are global. However, the alignment lines in a given layout may have different qualities. Some of them would be invalid if all alignment lines are detected under the same configuration. Therefore, the clustering algorithm should be adaptive to each alignment line.

RANSAC-based approach. Our solution is a variant of the classic *RANSAC* algorithm [37]. Since the number of edges is usually not big (from dozens to hundreds), it is possible to find globally optimal lines without randomness. Specifically, for every edge, we use the following iterative procedure to find a candidate alignment line: 1) the current edge as an initial inlier gives an initial fitted line; 2) all other edges are tested against the fitted line to find an updated set of inliers; 3) refit the line to the updated set of inliers. Steps 2 and 3 are repeated until convergence. Each edge as the initial inlier gives a candidate alignment line, and we pick the best line as the determinate alignment line. The inliers (edges) of the picked line are then removed from the current set of edges and we repeat the above iterative procedure to find the next alignment line till no more lines can be found. See the appendix for more details of Steps 2 and 3.

Once we find a candidate alignment line for each edge, we select the best alignment line $\mathcal{P}_a$. Generally we prefer an alignment line with a larger group of inliers and a lower variance of the vertical coordinate among the inliers. All the inlier edges corresponding to the selected alignment

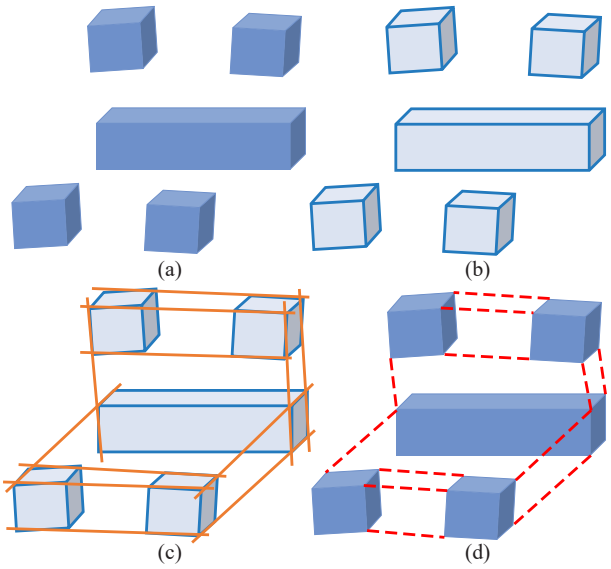


Fig. 6. Illustration for inferring relationships (dashed lines) for alignment of 3D elements.

line have edge-alignment relationships between each other (Figure 5 (d) and Figure 6 (d)).

Our algorithm is able to detect the edge alignment in any direction. In our implementation, we only detect the horizontal and vertical alignments for 2D elements, based on the observation that most 2D layouts are axis-aligned. For 3D elements, we detect the alignments with general directions and allow the elements to be in any orientations.

Inferring equal-spacing relationships. In our implementation, we allow the existence of equal-spacing relationships only in a group of graphic elements that have edge-alignment relationships. We use an iterative bottom-up clustering approach to detect equal-spacing relationships. The spacing is defined as the distance between every pair of *adjacent* elements. Initially, each spacing cluster consists of only one spacing. Each iteration groups two clusters with the smallest distance. The distance between two spacings d_1 and d_2 is defined as $|d_1 - d_2|/(d_1 + d_2)$. For two spacing clusters, it is defined as the difference of the average spacing between two clusters. The clustering process continues until the distance of the best pair of spacing groups is larger than a predefined threshold (we set as 0.15). Note that our gestural interface allows interactively adding equal-spacing constraints to non-adjacent pairs of elements (see an example in Figure 2 (c)).

4.2 Layout Refinement by Optimization

Following the previous beautification work [3], we use an optimization-based approach to refine the layout of 2D or 3D elements to satisfy the inferred relationships while retaining as much as possible the perceived 2D layout, i.e., the layout composed of 2D elements or projected 3D elements in the screen space. The optimization can be generalized as below.

Objective. Retaining the perceived layout can be achieved by retaining the position, orientation, and size of all elements in the screen space. For each graphic element $g \in \mathcal{G}$, where $\mathcal{G}$ is the set of input graphic elements, its position, orientation, and size are represented by vector

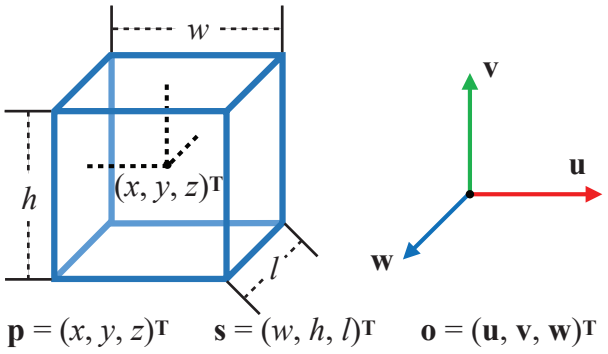


Fig. 7. Illustration of the variables during the optimization.

$\mathbf{p}_g$, $\mathbf{o}_g$, and $\mathbf{s}_g$, respectively (See Figure 7). $\mathbf{p}_g$ is a vector which contains the coordinates of the element center, e.g., $(x_g, y_g, z_g)^T$ for 3D elements. $\mathbf{o}_g$ is a vector that contains all the axis directions of an element, e.g., $(\mathbf{u}_g, \mathbf{v}_g, \mathbf{w}_g)^T$ for 3D elements. $\mathbf{s}_g$ is a vector which contains the size parameters, e.g., $(w_g, h_g, l_g)^T$ for 3D elements. Therefore, to retain the perceived layout we minimize the following objective function:

$$E = \sum_{g \in \mathcal{G}} (||\mathbf{M}_p(\mathbf{p}'_g - \mathbf{p}_g)||^2 + \alpha||\mathbf{o}'_g - \mathbf{o}_g||^2 + \beta||\mathbf{s}'_g - \mathbf{s}_g||^2), \tag{1}$$

where $\mathbf{p}'_g$, $\mathbf{o}'_g$, and $\mathbf{s}'_g$ are the variables being optimized, denoting the changed position, orientation and size of elements after refinement, respectively. α and β are used to control the relative weight of different objectives. We set $\alpha = 1$ and $\beta = 1$ in our implementation. $\mathbf{M}_p$ is the transformation matrix between the actual position and the perceived position. For 2D elements, $\mathbf{M}_p$ is the identity matrix since the actual position and perceived position are the same. For 3D elements, $\mathbf{M}_p$ is determined by the camera view when performing the layout editing. Therefore, Eq. (1) can be rewritten as:

$$\begin{aligned} E_{2D} &= \sum_{g \in \mathcal{G}_{2D}} (||\mathbf{p}'_g - \mathbf{p}_g||^2 + \alpha||\mathbf{o}'_g - \mathbf{o}_g||^2 + \beta||\mathbf{s}'_g - \mathbf{s}_g||^2), \\ E_{3D} &= \sum_{g \in \mathcal{G}_{3D}} (||\mathbf{M}_p(\mathbf{p}'_g - \mathbf{p}_g)||^2 + \alpha||\mathbf{o}'_g - \mathbf{o}_g||^2 + \beta|\mathbf{s}'_g - \mathbf{s}_g||^2). \end{aligned} \tag{2}$$

Here minimizing $||\mathbf{M}_p(\mathbf{p}'_g - \mathbf{p}_g)||^2$ is not equivalent to minimizing $||\mathbf{p}'_g - \mathbf{p}_g||^2$, since the transformation is not reversible.

Constraints. The above minimization is subject to a set of constraints. We first identify the intrinsic constraints on the axis directions of each graphic element g. For 3D elements, the constraints are listed below:

$$\begin{aligned} &\mathbf{u}_g\mathbf{v}_g = 0, \quad &&\mathbf{v}_g\mathbf{w}_g = 0, \quad &&\mathbf{w}_g\mathbf{u}_g = 0, \\ &||\mathbf{u}_g||^2 = 1, \quad &&||\mathbf{v}_g||^2 = 1, \quad &&||\mathbf{w}_g||^2 = 1. \end{aligned} \tag{3}$$

The above equations mean that the axis directions of an element should be unit vectors and orthogonal to each other. The intrinsic constraints on axis direction for 2D elements can be obtained similarly. If the aspect ratio a_g needs to be retained during the refinement (e.g., arranging 2D images), a constraint on size $\mathbf{s}_g = (w_g, h_g)^T$ is added, where $w_g = a_g h_g$.

The rest of the constraints are derived from the detected relationships. For alignment relationships, let $\mathcal{S}_a$ denote a

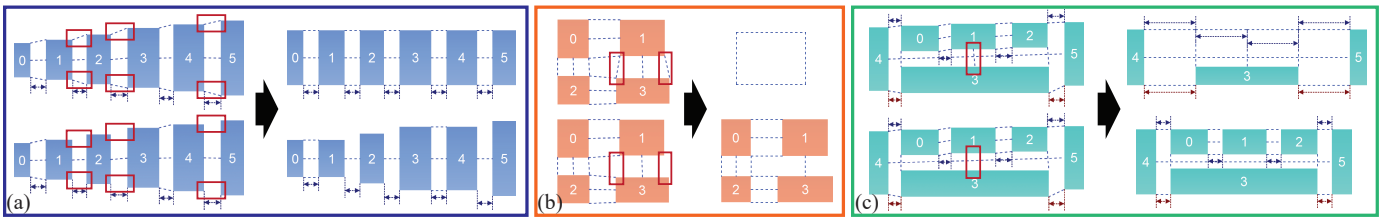


Fig. 8. Left in each box: input layouts. Right in each box: results by simultaneously enforcing all detected constraints (top) and results by automatically rejecting the highlighted constraints (bottom). Note that the constraint of keeping the aspect ratios of elements is enforced in (b) and (c) but not (a).

set of inliers that will be aligned to the same line $\mathcal{P}_a$. For each edge $s_i \in \mathcal{S}_a$, it has two properties, i.e., position $\mathbf{p}(s_i)$and direction $\mathbf{d}(s_i)$. These two properties are represented by the position $\mathbf{p}_g$, orientation $\mathbf{o}_g$, and size $\mathbf{s}_g$ of element g, which contains this edge. For example, for the top edge s_i of a 2D element g, we have $\mathbf{p}(s_i) = \mathbf{p}_g + \frac{1}{2}h_g\mathbf{v}_g$ and $\mathbf{d}(s_i) = \mathbf{u}_g$. If edges s_i and s_j have an edge-alignment relationship, we have the following constraints:

$$\begin{aligned} &\mathbf{d}(s_i) = \mathbf{d}(s_j), \\ &(\mathbf{p}(s_i) - \mathbf{p}(s_j)) \times \mathbf{d}(s_i) = 0. \end{aligned} \tag{4}$$

Each pair of neighboring graphic elements in $\mathcal{P}_a$ gives rise to a set of constraints in Eq. (4), resulting in a total of $|\mathcal{S}_a| - 1$ sets of constraints, where $|\mathcal{S}_a|$ is the cardinality of $\mathcal{S}_a$. For equal-spacing relationships, let $\mathcal{S}_s$ be a set of element pairs with the equal-spacing relationship detected for every element pair. The spacing between a pair of elements is represented by the positions, orientations, and sizes of these two elements. Enforcing the equality of these spacings introduces another $|\mathcal{S}_s| - 1$ constraints. For the 3D case, we add extra position constraints for anchor elements, which introduces k equality constraints if there are k anchor elements.

Optimization. The objective function (2) and some of the constraints in (3) and (4) are quadratic, therefore the optimization is a quadratic programming problem with quadratic constraints. Since our constraints are not convex, we can not directly solve the problem efficiently. We noticed that the quadratic constraints are all about optimizing the orientation, therefore we adopt a two-step procedure to obtain an approximate optimum. We first extract the orientation relations among elements, and optimize them using an iterative approach, i.e., iteratively changing the orientation of each elements until convergence. After determining the orientations, the remaining constraints are all linear, and thus the optimization problem can be solved efficiently.

In general, only a subset of the constraints can be enforced during the optimization. This is because simultaneously enforcing some constraints may cause the refined layout to deviate too much from the original layout (Figure 8 (a)). Worse, the layout may become degenerate in that the sizes of some graphic elements may become zero or negative (Figure 8 (b) and (c)).

Similar to [3], [4], we use an iterative approach to determine valid constraints. Specifically, we iteratively check the validity of each of our orientation, alignment, and spacing equality constraints by examining the optimized layout. After adding each constraint to the system, we check the current optimal solution and drop the constraint if the layout becomes degenerate or changes too much. Since our two-step optimization procedure ensures that the constraints are in the linear equality form, and the constraints are added incrementally, this strategy works efficiently. The alignment constraints are added in the order in which the corresponding alignment lines are detected, i.e., the constraints are preserved in a higher priority if the corresponding alignment line is detected earlier. Within the edge-alignment constraints corresponding to a specific alignment line, we rank them by their alignment error. This means that a constraint $(\mathbf{p}(s_i) - \mathbf{p}(s_j)) \times \mathbf{d}(s_i) = 0$ has a higher priority if $|(\mathbf{p}(s_i) - \mathbf{p}(s_j)) \times \mathbf{d}(s_i)|$ is smaller. Note that the orientation constraint between a pair of elements is realized right before adding their alignment constraint, therefore $\mathbf{d}(s_i)$ is determined. After considering all the constraints of an edge-alignment pattern, we add the detected equal-spacing relationships from this alignment pattern. All constraints of an equal-spacing pattern are simultaneously added, and they are all rejected if the new layout becomes degenerate or deviates too much from the original layout, since equal-spacing often exists as a global pattern. We use the CHOLMOD sparse linear system solver [38] to incrementally solve the resulting constraint optimization problem after each constraint is added. It provides the fast update for the linear system, without repeatedly solving it from scratch, and thus is able to solve the incremental constraint optimization problem efficiently. During the optimization, the relative weights between different objectives are equal, i.e., β in Eq. (1) or (2) is 1, and α is not applicable due to our two-step optimization procedure.

Since the most popular 2D elements editors only support axis-aligned edge alignment and equal-spacing relationships, our current implementation did not consider orientation constraints for 2D elements, i.e., all the elements are axis-aligned and are not rotated during the editing. We included the orientation constraints in our 3D layout system. See also Figure 6 and the accompanying video.

While our simple strategy generally works well and efficiently, we are aware of another possible solution which determines constraints by adding linear inequality constraints to restrict the solution domain. For example, the refined size of a graphic element should be within a certain range. We can then incrementally add a constraint and retain it only if the feasible region of this constraint problem is not empty. For this solution, we may resort to existing constraint solvers like QOCA [35] and Gecode [36].

5 User Studies

We have extensively tested our technique on layouts of various patterns, most of which are time-consuming to produce when snapping and alignment commands are used. See

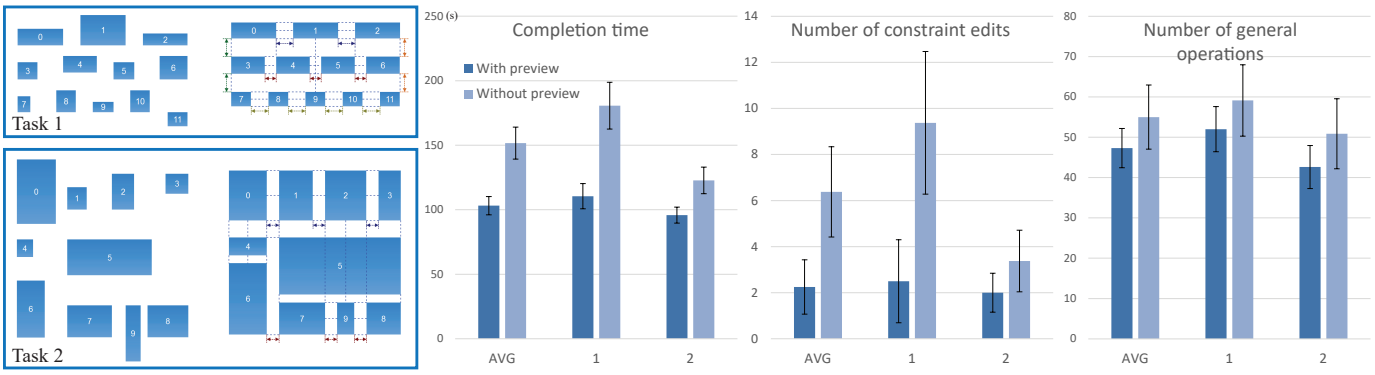


Fig. 9. Two layout refinement tasks used in study 1 and the resulting statistics. Error bars represent standard error of the mean.

some of the tested layouts in Figures 8–11. Our tool is also applicable to layouts with nested elements (See Figure 10). Note that a fixed set of parameter values were always used throughout our experimentation. Three user studies were conducted to evaluate the effectiveness of our technique. We focused on representative layouts and avoided introducing tasks with over-complicated layouts, since it would take a lot of time to complete such tasks with traditional tools, making the studies unnecessarily long. We will discuss the scalability of our interface when dealing with complicated tasks in Section 6.

5.1 Study 1: Evaluation of Beautification Preview

We first evaluated the performance of the beautifier with and without a preview window. This study was based on 2D layout tasks. Eight university students helped with the evaluation. They have extensive experience in using the traditional 2D alignment tools, e.g., in Microsoft PowerPoint.

Apparatus. The study was conducted on an ordinary PC (DELL Optiplex 960), with 3.00 GHz Intel Core 2 Duo CPU and 4.00 GB RAM. Two LCD monitors (20-inch and 17-inch) were connected, one for the beautifier interface (with the preview window if used), and the other only for displaying a static target layout. Our beautifier achieved real-time performance for moderately complex layouts. For example, for a typical layout containing a dozen of elements like those shown in Figure 11, it took our technique less than 0.01 seconds for alignment relationships inference and less than 0.02 seconds for layout refinement.

Tasks. We asked participants to perform 2D layout refinement. For each task, each participant was asked to refine a source layout, which already resembled a target layout with visualized alignment and equal-spacing relationships (Figure 9), displayed on a separate monitor. The source and target layouts were consistently numbered so that the participants knew the correspondence between the elements. Common operations, such as element translation, resizing and selection (by single click or rectangle selection tool), were allowed. Element creation and removal are disabled.

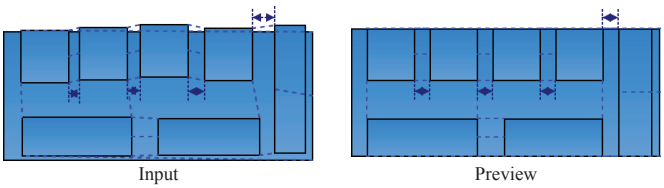


Fig. 10. The application of our tool to layouts with nested elements.

Two tasks (Figure 9) were tested and had different layout complexity. Task 1 was more challenging since it involved more potential ambiguities during refinement. Each participant was required to complete each of the two tasks, with two tools: our beautifier with and without a preview window. For the latter, the beautified result was instantly applied to the input layout, as similarly done in the previous works [2], [23]. See the accompanying video for such an interface in action. In total, we had 8 (participants) $\times$ 2 (tools) $\times$ 2 (tasks) $=$ 32 trials.

With each of the two tools (i.e., the beautifier with and without a preview), the participants were asked to quickly reproduce each of the two target layouts by achieving as many visualized target relationships as possible. However, they were allowed to proceed to the next task without fully reproducing the target layout of the current task. To help the participants better track their progress, the alignment and equal-spacing relationships of the layout were visualized and shown in the preview window or the input window in the case of no preview window. The order of the tasks and the tools in each task were counter-balanced across participants. Before the study, the participants were introduced to these two tools, and they practiced in a short warm-up session until they felt comfortable. The whole study lasted less than 20 minutes on average for each participant.

Performance measures. During the study, the following information was recorded for quantitative analysis: the completion time of individual trials, the time spent on element editing (selection, moving, resizing), the number of general operations (moving, resizing, duplicate, removal, undo), and the number of edits for edge-alignment constraints and equal-spacing constraints.

Results. Figure 9 plots the statistics of the core information captured in Study 1. The preview window significantly shortened the average completion time per task among the participants, from 151.7 seconds to 103.2 seconds. A significant difference was confirmed by repeated measures analysis of variance (repeated measures ANOVA): $F = 57.77$ and $p = 0.000126$. The benefit of our interface with the preview window was even clearer for Task 1, a more challenging task. The average completion time of Task 1 was 110.5 seconds with our interface, compared to 180.6 seconds without the preview window ($F = 41.59$, $p = 0.000351$).

Figure 9 also shows the average number of operations for

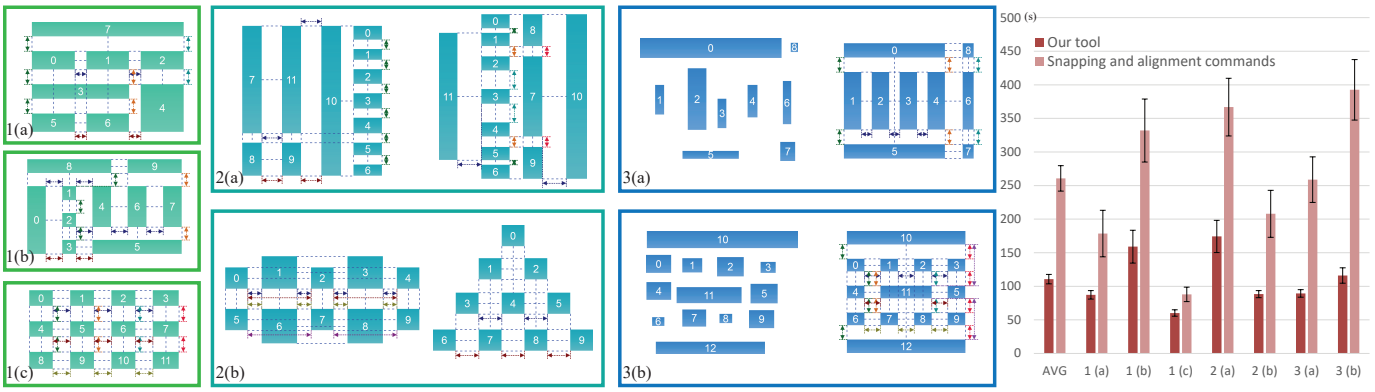


Fig. 11. From left to right: 1(a)-(c): target layouts used in the layout creation scenario; 2(a)&(b): source layouts (left) and target layouts (right) in the layout editing scenario; 3(a)&(b): source (left) and target layouts (right) in the layout refinement scenario; average completion time (in seconds) of each task. Error bars represent standard error of the mean.

editing constraints and the average number of general operations (moving, resizing, etc). While on average the number of general operations was less with our interface, the difference was not statistically significant. Repeated measures ANOVA confirmed a significant difference in the average of constraint edits for Task 1 ($F = 17.78$, $p = 0.003954$) but not for Task 2 ($F = 2.951$, $p = 0.12951$) between the beautifier with and without the preview window. For the more challenging Task 1, without the preview window, the users easily got distracted by intermediate beautification results, making them difficult to achieve desired layouts simply by direct repositioning and resizing of elements. Thus they had to resort to the tools for editing constraints more often. It is interesting to note that the standard errors here were relatively large, indicating that different participants might have different preferences for the interface for editing constraints.

5.2 Study 2: Comparing to Snapping & Alignment Commands

We conducted a user study to evaluate the effectiveness of our technique for 2D layout tasks, compared to snapping and alignment commands, which are arguably the most popular tools available in almost all commercial 2D graphic editors like OmniGraffle, Visio, and InDesign. We implemented these standard tools in a way like in PowerPoint 2013. Specifically, snapping was activated only for the element being dragged. This element could be snapped to achieve edge/center alignment or equal spacing with other elements. The compared system supported 8 commands: align top/middle/bottom/left/center/right, and distribute horizontally/vertically. We chose the lead object for alignment similar to PowerPoint, e.g., the topmost element for top-align, and the outermost elements for even distribution. We deliberately excluded layout helper features beyond the standard snapping and alignment commands, since it is expected that such extra functions would similarly benefit our tool and the compared system.

Apparatus and participants. We used the same set of instruments as those in Study 1. Another 11 university students were recruited for the user study. Again all of them had used traditional alignment tools extensively. A handful of them were even good at vector graphics editing and were familiar with professional software like Adobe Illustrator.

Tasks. Besides layout refinement, we evaluated the performance of the traditional tools and ours in another two scenarios, i.e., creating and editing layouts. In the creation scenario, three tasks were tested. For each task, each participant was asked to create a layout from scratch towards a reference target layout (Figure 11 1(a)-(c)). The participants were allowed to draw, duplicate, or remove elements. The layout editing scenario comprises two tasks, each of which required the participants to significantly change an input layout towards a target layout (Figure 11 2(a)-(b)). Element creation and removal were disabled. Like Study 1, the participants were asked to complete two layout refinement tasks (Figure 11 3(a)-(b)). But this time we locked the aspect ratios of the elements, controlled by a toggle button. For the editing and refining scenarios, consistent correspondence numbers were provided.

In total, there were seven tasks under the three scenarios. All target layouts in these tasks had reasonable complexities to simulate the layout creation tasks in practice while maintaining an acceptable duration of the study. Each participant was asked to complete each of them twice, one with our tool and the other with the set of traditional alignment tools. That is, our experiment involved 11 (participants) $\times$ 2 (tools) $\times$ 7 (tasks) $=$ 154 trials. The tools in each task were (almost) counter-balanced and the order of the tasks in each scenario was random. The participants took breaks between different scenarios so that they were briefed on the newly enabled or disabled operations before each scenario. For simplicity, no break was allowed between tasks in each scenario. The whole study lasted around one hour on average for each participant.

Performance measures. Besides the measures used in Study 1, the following information was also recorded: the time spent on drawing the elements (for the creation scenario only), and the number of specific operations with the traditional tools (commands like edge alignment and equal spacing, etc).

Results. Figure 11 shows the target layouts in each task, and the corresponding average completion times. Repeated measures ANOVA found a significant difference in the average completion time per task among the participants between our tool and the traditional tools ($F = 78.8$, $p < 4 \times 10^{-6}$). On average, significantly less time was needed to accomplish each task using our tool (110.7 sec-

onds) compared to the traditional tools (260.7 seconds).

Our tool performed much better than the traditional tools for target layouts involving complex relationships such as Tasks 1(b) and 2(a). For easier tasks, such as Tasks 1(a) and 1(c), using our tool was still faster. For the tasks in the refinement scenario, due to the locked aspect ratios of the elements, the editing freedom was seriously restricted, making such tasks more challenging to complete for the traditional tools. More than half of the participants failed to reproduce the target layouts with the traditional tools, while all of them had no difficulty with our tool. Given the above results, we believe that the superiority of our tool would hold, and even become more notable, when the complexities of target layouts increase.

In Figure 12 we show more statistics from the user study. There was no statistically significant difference in the drawing time between the two tools ($p = 0.465$), though we observed that our tool was still slightly faster. Our tool required significantly fewer general operations ($p < 5 \times 10^{-5}$). The task success rates also reflect the effectiveness of our tool. For almost all the tasks, each participant could successfully reproduce the target layouts with our tool. In contrast, the completion rates for the traditional tools were much lower, despite they had been used by all the participants on a regular basis. This confirms the ease of use of our tool, even for the first-time users.

Figure 12 also shows the average number of edits on edge alignment and equal-spacing constraints for individual tasks. While the number of edits was small, constraint editing was performed for 5 out of 7 tasks. We speculate that the number of constraint edits needed is correlated with layout complexity, which is somewhat reflected by the (average) completion time. For example, the number of constraint edits used for Task 1(b) was much bigger than that for Task 1(a) and Task 1(c). A similar conclusion on the average completion time of these tasks could be reached.

It can be seen that the standard error of the mean (shown as error bars in Figures 11 and 12) of the completion time, editing time, and editing number is much larger with the traditional tools. This means that the performance of our tool was more consistent across all participants. In other words our tool was less dependent on the experience of the individual participants.

Our statistics show that on average each participant performed 0.97 edge-alignment commands and 1.90 equal-spacing commands per task. This is mainly because equal-spacing commands were more indispensable, while edge-alignment commands could be replaced with more intuitive snapping operations. This reiterates the fact that snapping has to be used together with command-based alignment tools in many cases.

We also got some interesting observations from the user study. With our tool, some participants tended to draw or edit elements carefully at the beginning despite being told that our tool can tolerate rough inputs. However, they quickly get used to our tool and performed the creation and editing operations more casually. Another observation was that, in the refining scenario, when using the traditional tools, some participants moved all the elements away before editing, though most of the input elements were already located very close to the target positions. Such behavior was

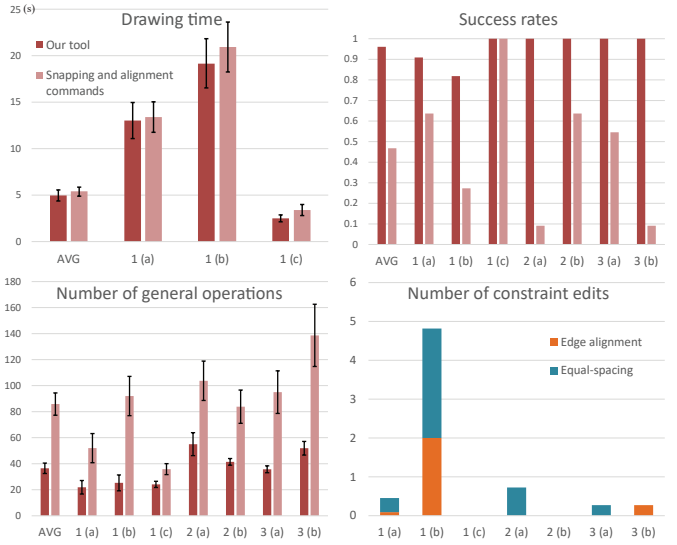


Fig. 12. More statistics. The timings are in seconds. Error bars represent standard error of the mean.

never observed with our tool. This behavior was largely due to the not-aligned elements triggering often excessive and distracting suggestions from snap-dragging.

After completing the tasks, we asked each participant to provide feedback on the two tools in terms of their ease of use. All participants except one preferred our tool. They expressed that our tool enables them to disregard the order of specific operations and focus on the layout design. One participant who preferred the traditional tools liked the full user control and disliked rough inputs. This concern could be addressed by integrating snap-dragging into our framework.

5.3 Study 3: Comparing to 3D Arrangement Interface

We evaluated our framework for 3D layout tasks, compared to the traditional multi-viewport interface, which is a standard component of 3D graphic editors like Autodesk 3ds Max. We implemented this traditional tool following the one in 3ds Max, i.e., by providing the top, front, left, and free viewports for editing elements. The user could freely change the position and/or size of 3D elements without affecting their depth in each viewport. To further increase the usability, we enabled the 2D snapping effect for alignment and equal-spacing in the top, front, and left viewports, except for the free viewport, since 2D snapping could not be directly extended to this viewport.

Apparatus. This study was conducted on another PC (Dell Precision Tower 7910), with 2.3 GHz Intel Xeon CPU and 64.00 GB RAM. Two 24-inch LCD monitors were connected for this study. Similar to Studies 1 and 2, these two monitors were used for displaying a target layout and the layout editor (with our interface or the traditional interface), respectively. Our beautifier for 3D layout optimization achieved real-time performance. We also tested it on the PC used in Studies 1 and 2, and the real-time performance was also guaranteed.

Participants. We recruited another 15 university students to help in the study. Six of them were familiar with a 3D graphic editor. Six participants had 3D elements manip-

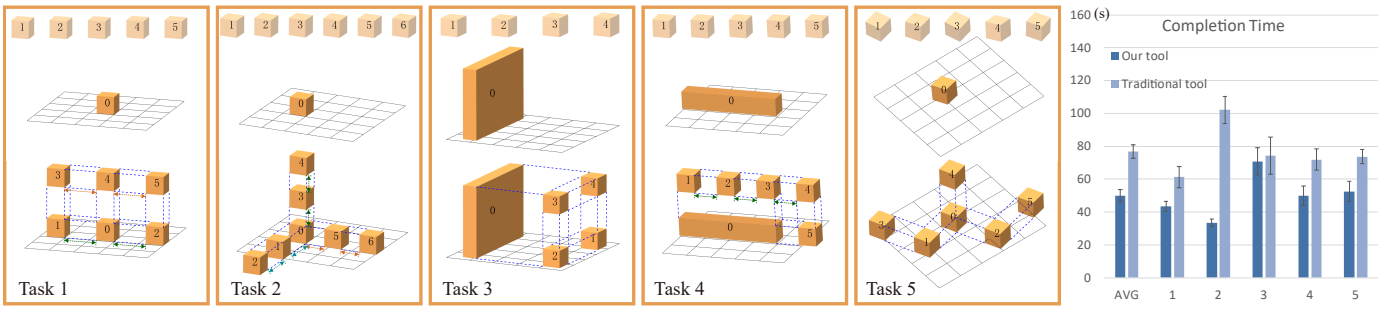


Fig. 13. Left: The target layouts used in Study 3. Right: The average completion time (in seconds) of each task. Error bars represent standard error of the mean.

ulation experiences. The other three participants considered them as novice users of a 3D graphic editor.

Tasks. We evaluated the performance of the traditional and our interfaces in creating 3D layouts with given 3D elements. This layout creation scenario is very common in 3D editing, since 3D elements are often reusable in assembling 3D objects or creating 3D scenes. As shown in Figure 13 (Left), five tasks were tested in this study. For each task, the participants were asked to achieve a target layout by arranging a set of elements, with the traditional and our interfaces respectively. To reduce the difficulty of using the traditional interface, we did not require the participants to perform element resizing operations. The target layouts were achievable by only element reposition operations. Among the given elements, one was placed in the center area of the scene and considered as the anchor element. Other elements were placed in the border area of the scene for further arrangement. When using our interface, the elements in the border area were considered as floating elements, i.e., elements without their 3D positions. In Tasks 1 – 4, the orientations of the elements before and after the arrangement were all aligned with the global axis, which is common in creating 3D objects or scenes. In Task 5, the original orientations of the elements had slight variations. In addition, their orientations in the target layout were not aligned with the global axis. It corresponded to more general 3D arrangement tasks, in which elements could be in arbitrary orientations. The order of two interfaces was (almost) counter-balanced and the tasks were tested with a random order. In total there were 5 (tasks) $\times$ 2 (interfaces) $\times$ 15 (participants) $=$ 150 trials. Before completing the tasks, there was a 10-minute practice session for the participants to get familiar with the traditional and our interfaces. There was no break between tasks in this study.

Performance measures. We recorded the following information for quantitative analysis: the completion time of individual trials, the number of general operations (moving, view transition, undo, and redo), and the number of relation editing with our interface.

Results. Figure 13 (Right) shows the resulting average completion time for each task. Figure 14 shows the other statistics. Repeated measures ANOVA confirmed that there was a significant difference in the average completion time ($F = 61.20$, $p < 1.8 \times 10^{-6}$) and the average number of general operations ($F = 127.27$, $p < 2.1 \times 10^{-8}$) among the participants between our and traditional interfaces. On average, the participants spent significantly less time (50.1s vs. 76.7s) and fewer general operations (8.8 vs. 20.4) in completing each task using our interface compared to the traditional interface. Compared with the tasks in Studies 1 and 2, these tasks could be completed with less time and fewer operations, since the participants could focus on element moving operations only. Figure 14 (Left) also shows the average number of relation edits when using our interface. It is worth noting that participants tended to use the gestural tools more frequently (on average, 3.29 constraint edits vs. 8.8 general operations), compared with the tasks in Studies 1 and 2. It is possibly because the alignment relations among 3D elements had more variations and ambiguities (see Figure 3). It was thus more difficult to achieve the desired alignment relations by element arrangement, compared with 2D cases. None of the participants used the gestural tool to edit the equal-spacing relations, since they could be easily formed as long as the desired alignment relations were achieved.

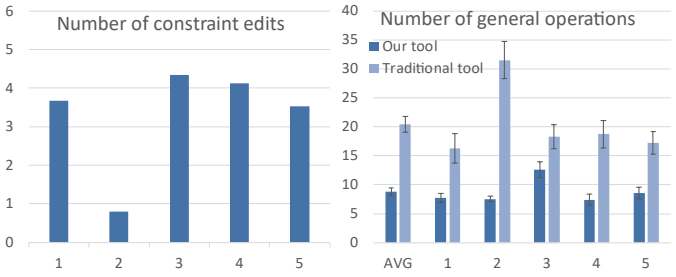


Fig. 14. More statistics. Error bars represent standard error of the mean.

The performance of our and the traditional interfaces varied for individual tasks. Task 1 was the simplest one with both interfaces. Confirmed by repeated measures ANOVA, the participants spent significantly less time with our interface ($p < 0.03$) since they could perform rough operations. The number of general operations was also significantly less when using our interface ($p < 0.002$). Task 2 was more complex than Task 1 when using the traditional interface since the elements in the target layout had varied depths. The participants needed to move the elements in the appropriate viewports to achieve the target layout. It confused the novice users who were not familiar with the multiple viewport interface. Our interface significantly outperformed the traditional one in both the completion time ($p < 8 \times 10^{-7}$) and the number of operations ($p < 2 \times 10^{-6}$). For Task 3, our interface was slightly faster but the difference between the two interfaces was not significant ($p = 0.6$). This task was a little hard with our interface since the relations were

occluded by each other. Some participants were confused during editing and thus spent more time on figuring out how to specify the desired relations. Still, our interface required significantly fewer general operations ($p < 0.04$). Task 4 was similar to Task 1, but required the participants to plan the operations carefully when using the traditional interface. In contrast, our interface was not affected by the order of the operations, and therefore achieved significantly less completion time ($p < 0.002$) and fewer general operations ($p < 8 \times 10^{-4}$). Task 5 was challenging when using the traditional interface. The user needed to first fine-tune the elements' orientations and performed a group transformation to make them aligned with the global axis. After the arrangement, the user needed to perform another group transformation to achieve the target layout. To reduce the difficulty when using the traditional interface, we assumed that the elements' orientations were already fine-tuned and aligned with the global axis. The participants did not need to perform group transformation after the arrangement either. When using our interface, we still used the original setting of Task 5. The results showed that our interface achieved significantly less completion time ($p < 0.007$) and fewer general operations ($p < 9 \times 10^{-4}$) than the traditional interface, even though the traditional interface was used in a simplified setting.

We also obtained some feedback from the participants. They expressed that our interface was easy to use, and effectively saved the operation time of specifying the depths of the elements. The visualization was very useful for perceiving the relations among elements. They also commented that the interface was not suitable for arranging elements with occlusions, in which case the relation visualization might introduce severe visual clutter, preventing them from achieving desired layouts.

6 Discussions

To use our tool in existing graphic editors like PowerPoint, the user may first select a set of target elements and then enter a "layout beautification mode", for example, via an activation button, menu or gesture. Similar to the interface described in our prototype, we may pop up a dialog to edit elements and/or constraints while previewing the beautified layout in the input panel. In the mode of element editing, our interface is completely compatible with existing layout helper features. All edits are confirmed after the user hits the "apply" button; otherwise, they are canceled.

While the results of Study 1 imply the usefulness of having a preview of the refined layout in the experimental condition, rendering the original and refined layouts in separate windows might cause the problem of divided attention. To alleviate this problem we might use a smaller preview window instead of a full-scale copy of the design. It is not necessary to always activate the preview window especially when the input layout is undergoing dramatic changes, e.g., at the beginning of layout editing and creation tasks in Study 2, and the tasks in Study 3.

To further solve the problem of divided attention and to save space required by an extra preview window, we could show the preview in place. For example, we might overlay the beautified layout as a translucent layer on top of the original layout. Since this solution easily results in visual clutter, it might work well for only relatively simple layouts with fewer elements, but would likely not be able to scale to very complex diagrams. In the future, it is worth evaluating whether such alternative solutions would be more effective than a simple preview window in our current prototype.

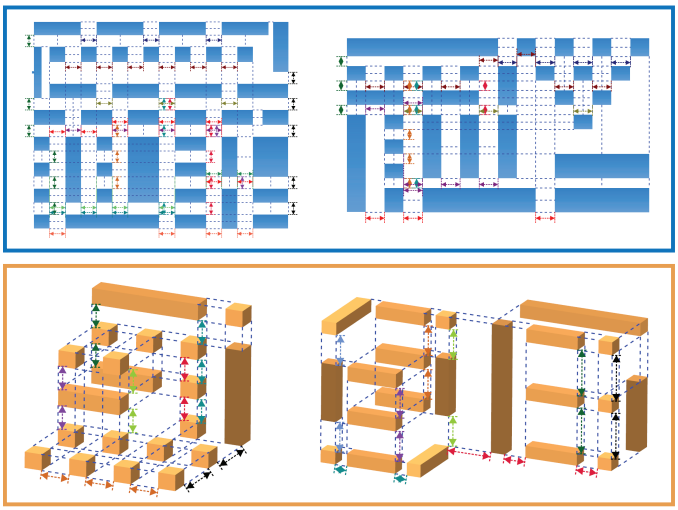


Fig. 15. Our interface has good scalability when the complexity of the layouts increases. For practiced users, it took less than 4 minutes to finish each 2D task (Top) and less than 5 minutes to finish each 3D task (Bottom).

Our interface has good scalability with the complexity of layouts. To create each 2D or 3D layout shown in Figure 15, it took less than 5 minutes on average for practiced users. Compared to the completion times reported in Studies 2 and 3, we conclude that the interaction time is roughly proportional to the number of elements and relations in a target layout, indicating good scalability of our interface. The real-time performance was not affected, thanks to the high efficiency of our algorithm. Although our interface is able to handle dozens of elements for practiced users, it is not wise to edit too many elements simultaneously for novice users, since visualizing too many constraints may cause visual clutter and thus confusion. This issue is more notable in 3D applications due to the overlap between elements and visualized constraints. It is thus advisable to utilize our interface in an incremental way when dealing with a large number of elements, i.e., decomposing the whole layout into several sub-layouts, and creating them one by one.

Our current implementation only considers the edges of the bounding boxes of the elements when detecting relations. Although this setting is sufficient for most layout creation tasks, it would be promising to include a tool for manually adding edges to the elements for relation detection. The manually added edges would help to form more complex patterns, and therefore further strengthen the layout creation capability of our interface.

Since our main goal in this work is to find a better alternative to snapping and alignment commands, we focused only on a limited set of simple constraints, i.e., alignment and equal spacing. However, it is easy to incorporate new constraints, such as symmetry and length equality into our framework. Similarly, although our current prototype concentrated on grid-like layouts, it is possible to handle more general layouts by introducing advanced pattern de-

tectors, e.g., to detect elements distributed roughly along an arbitrary line, circle, etc.

7 Conclusion and Future Work

We presented a novel user interface for layout beautification. Our user studies confirmed that our technique is very effective and can even be used as a better replacement of the standard snapping and command-based alignment tools, given its faster performance and better ease of use. We speculate that the advantages of our tool would be even clearer to novice users who have little or no experience in creating precise layouts.

It would be interesting to apply our tool to real-world examples like posters and to really integrate it with existing graphic editors to examine the interplay between our tool and other layout helper features. Our current implementation requires explicit mode switching to activate or deactivate the gesture interface for editing geometric constraints. In the future, we will explore a modeless interface where the user could use the editing operations of both element and constraint without explicit mode switching.

Acknowledgments

We thank the reviewers for their constructive comments and the user study participants for their time. This work was supported in parts by NSFC (61602310, 61761146002, 61861130365), GD Higher Education Innovation Key Program (2018KZDXM058), GD Science and Technology Program (2015A030312015), Shenzhen Innovation Program (JCYJ20170302154106666), LHTD (20170003), the Research Grants Council of HKSAR (HKUST16210718), a Gift from Adobe, the Center for Applied Computing and Interactive Media (ACIM) of School of Creative Media, CityU, and the National Engineering Laboratory for Big Data System Computing Technology, SZU.

References

[1] E. A. Bier and M. C. Stone, "Snap-dragging," in *ACM SIGGRAPH Computer Graphics*, vol. 20, no. 4, 1986, pp. 233–240.

[2] D. Galindo and C. Faure, "Perceptually-based representation of network diagrams," in *International Conference on Document Analysis and Recognition*, vol. 1, 1997, pp. 352–356.

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

[4] T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka, "Interactive beautification: a technique for rapid geometric design," in *UIST '97*, 1997, pp. 105–114.

[5] T. Igarashi and J. F. Hughes, "A suggestive interface for 3D drawing," in *UIST '01*, 2001, pp. 173–181.

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

[7] R. Raisamo and K.-J. Räihä, "A new direct manipulation technique for aligning objects in drawing programs," in *UIST*, 1996, pp. 157–164.

[8] 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*. ACM, 2015, pp. 2787–2795.

[9] P. Baudisch, E. Cutrell, K. Hinckley, and A. Eversole, "Snap-and-go: helping users align objects without the modality of traditional snapping," in *CHI*, 2005, pp. 301–310.

[10] E. A. Bier, "Snap-dragging in three dimensions," in *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 2. ACM, 1990, pp. 193–204.

[11] B. Nuernberger, E. Ofek, H. Benko, and A. D. Wilson, "Snaptoreality: Aligning augmented reality to the real world," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 1233–1244.

[12] M. Frisch, S. Kleinau, R. Langner, and R. Dachselt, "Grids & guides: multi-touch layout and alignment tools," in *CHI '11*, 2011, pp. 1615–1618.

[13] B. Reinert, T. Ritschel, and H.-P. Seidel, "Interactive by-example design of artistic packing layouts," *ACM Trans. Graph.*, vol. 31, no. 6, 2013.

[14] K. Ryall, J. Marks, and S. Shieber, "An interactive constraint-based system for drawing graphs," in *UIST '97*, 1997, pp. 97–104.

[15] I. Tollis, P. Eades, G. Di Battista, and L. Tollis, *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall New York, 1998, vol. 1.

[16] 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.

[17] C. Alvarado, "Sketch recognition user interfaces: Guidelines for design and development," in *Proceedings of AAAI Fall Symposium on Intelligent Pen-based Interfaces*, vol. 1, no. 2, 2004.

[18] J. Arvo and K. Novins, "Fluid sketches: continuous recognition and morphing of simple hand-drawn shapes," in *UIST '00*, 2000, pp. 73–80.

[19] T. Hammond and R. Davis, "Automatically transforming symbolic shape descriptions for use in sketch recognition," in *AAAI '04*, ser. AAAI'04. AAAI Press, 2004, pp. 450–456. [Online]. Available: http://dl.acm.org/citation.cfm?id=1597148.1597222

[20] B. Paulson and T. Hammond, "PaleoSketch: accurate primitive sketch recognition and beautification," in *IUI '08*, 2008, pp. 1–10.

[21] R. C. Zeleznik, A. Bragdon, C.-C. Liu, and A. Forsberg, "Lineogrammer: creating diagrams by drawing," in *UIST '08*, 2008, pp. 161–170.

[22] C. L. Zitnick, "Handwriting beautification using token means," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 53:1–53:8, 2013. [Online]. Available: http://doi.acm.org/10.1145/2461912.2461985

[23] D. Bolz, "Some aspects of the user interface of a knowledge based beautifier for drawings," in *IUI '93*, 1993, pp. 45–52. [Online]. Available: http://doi.acm.org/10.1145/169891.169897

[24] S. Cheema, S. Gulwani, and J. LaViola, "Quickdraw: Improving drawing experience for geometric diagrams," in *CHI '12*, 2012, pp. 1037–1064. [Online]. Available: http://doi.acm.org/10.1145/2207676.2208550

[25] H. Jiang, L. Nan, D.-M. Yan, W. Dong, X. Zhang, and P. Wonka, "Automatic constraint detection for 2d layout regularization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 8, pp. 1933–1944, 2015.

[26] O. Veselova and R. Davis, "Perceptually based learning of shape descriptions for sketch recognition," in *AAAI '04*, 2004. [Online]. Available: http://doi.acm.org/10.1145/1185657.1185789

[27] T. Hammond and R. Davis, "Interactive learning of structural shape descriptions from automatically generated near-miss examples," in *IUI '06*, 2006, pp. 210–217. [Online]. Available: http://doi.acm.org/10.1145/1111449.1111495

[28] T. Dwyer, K. Marriott, and M. Wybrow, "Dunnart: A constraint-based network diagram authoring tool," in *Graph Drawing*, 2009, pp. 420–431.

[29] M. Gleicher and A. Witkin, "Drawing with constraints," *The Visual Computer*, vol. 11, no. 1, pp. 39–51, 1994.

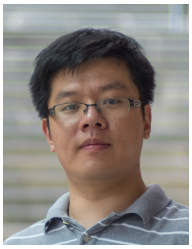
[30] I. E. Sutherland, "Sketch pad a man-machine graphical communication system," in *Proceedings of the SHARE design automation workshop*, 1964, pp. 6–329.

[31] M. Wybrow, K. Marriott, L. Mciver, and P. J. Stuckey, "Comparing usability of one-way and multi-way constraints for diagram editing," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 14, no. 4, p. 19, 2008.

[32] C. Zeidler, C. Lutteroth, W. Sturzlinger, and G. Weber, "The auckland layout editor: an improved gui layout specification process," in *UIST '13*, 2013, pp. 343–352.

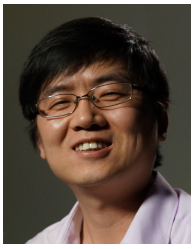
[33] G. J. Badros, A. Borning, and P. J. Stuckey, "The cassowary linear arithmetic constraint solving algorithm," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 8, no. 4, pp. 267–306, 2001.

[34] H. Hosobe, "A modular geometric constraint solver for user interface applications," in *UIST '01*, 2001, pp. 91–100.
[35] K. Marriott and S. S. Chok, "Qoca: A constraint solving toolkit for interactive graphical applications," *Constraints*, vol. 7, no. 3-4, pp. 229–254, 2002.
[36] C. Schulte, M. Lagerkvist, and G. Tack, "Gecode," *Software download and online material at the website: http://www. gecode. org*, 2006.
[37] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.
[38] T. Davis, "Cholmod: a sparse supernodal cholesky factorization package., version 2.1.2," 2013, university of Florida, Available online at http://www.cise.ufl.edu/research/sparse/cholmod/.



Pengfei Xu is an Assistant Professor of College of Computer Science and Software Engineering at Shenzhen University. He received his Bachelor degree in Math from Zhejiang University, China, in 2009 and his Ph.D. degree in Computer Science from Hong Kong University of Science and Technology in 2015. His primary research lies in Human Computer Interaction and Computer Graphics.

Guohang Yan is a master student with the Visual Computing Research Center at College of Computer Science and Software Engineering, Shenzhen University. He received his Bachelor degree from Pingdingshan University in 2016. His research interest is in Computer Graphics.



Hongbo Fu received a BS degree in information sciences from Peking University, China, in 2002 and a PhD degree in computer science from the Hong Kong University of Science and Technology in 2007. He is an Associate Professor at the School of Creative Media, City University of Hong Kong. His primary research interests fall in the fields of computer graphics and human computer interaction. He has served as an Associate Editor of The Visual Computer, Computers & Graphics, and Computer Graphics Forum.

Takeo Igarashi is a Professor of Computer Science Department at The University of Tokyo. He received a Ph.D from the Department of Information Engineering at The University of Tokyo in 2000. He then worked as a post doctoral research associate at Brown University (2000 - 2002). He joined the University of Tokyo as an Assistant Professor in 2002, and became a Professor in 2011. His research interest is in user interfaces and interactive computer graphics.

Chiew-Lan Tai is a Professor of Computer Science at the Hong Kong University of Science and Technology. She received the BSc degree in mathematics from the University of Malaya, MSc in computer & information sciences from the National University of Singapore, and DSc degree in information science from the University of Tokyo. Her research interests include geometry processing, computer graphics, and interaction techniques.

Hui Huang is a Distinguished Professor of Shenzhen University, where she directs the Visual Computing Research Center. She received her PhD in Applied Math from The University of British Columbia in 2008 and another PhD in Computational Math from Wuhan University in 2006. Her research interests span on Computer Graphics, Vision and Scientific Computing. She is currently an Associate Editor-in-Chief of The Visual Computer, a Senior Member of IEEE and ACM, and a Distinguished Member of CCF.