SweepCanvas: Sketch-based 3D Prototyping on an RGB-D Image

Yuwei Li
ShanghaiTech University
Shanghai, China
liyw@shanghaitech.edu.cn

Xi Luo
ShanghaiTech University
Shanghai, China
luoxi@shanghaitech.edu.cn

Youyi Zheng*
State Key Lab of CAD&CG,
Zhejiang University
Hangzhou, China
zyy@cad.zju.edu.cn

Pengfei Xu
Shenzhen University
Shenzhen, China
pengfeixu.cg@gmail.com

Hongbo Fu
City University of Hong Kong
Hong Kong
hongbofu@cityu.edu.hk



Figure 1. Our *SweepCanvas* system allows artists to quickly create 3D prototype models using a sketch-based interface on top of an RGB-D image as context. The left and right examples were created within 30 and 60 minutes, respectively, by a user with a short training period. Left and middle right are the original images overlaid with 3D models.

ABSTRACT

The creation of 3D contents still remains one of the most crucial problems for the emerging applications such as 3D printing and Augmented Reality. In Augmented Reality, how to create virtual contents that seamlessly overlay with the real environment is a key problem for human-computer interaction and many subsequent applications. In this paper, we present a sketch-based interactive tool, which we term *SweepCanvas*, for rapid exploratory 3D modeling on top of an RGB-D image. Our aim is to offer end-users a simple yet efficient way to quickly create 3D models on an image. We develop a novel sketch-based modeling interface, which takes a pair of user strokes as input and instantly generates a curved 3D surface by sweeping one stroke along the other. A key enabler of our system is an optimization procedure that extracts pairs of spatial planes from the context to position and sweep the strokes. We demonstrate the effectiveness and power of our modeling system on various RGB-D data sets and validate the use cases via a pilot study.

Author Keywords

Sketch-based modeling, modeling in context, swept surfaces

ACM Classification Keywords

H.5.2 [Information Interfaces and Presentation]: User Interfaces - Interaction styles, User-centered design; I.3.8 [Computer Graphics]: Application

INTRODUCTION

3D modeling is an important research topic in both the graphics and HCI fields. The well-adopted 3D modeling tools like Maya and Blender support powerful and general-purpose 3D modeling from scratch. However, the modeling process often requires tedious labor and professional skills. On the other hand, there is a high demand for 3D modeling tools geared towards normal users, in particular for the emerging applications of 3D printing and augmented reality (AR). One of the common tasks here is to design 3D virtual objects that can tightly or loosely fit into the real world. The traditional 3D modeling software becomes even more awkward for such tasks, since the contextual information of the real world is ignored during the modeling process and thus proper interaction between virtual and real-world objects have to be achieved in additional steps.

*corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

UIST'17, Oct. 22–25, 2017, Quebec City, Canada

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 123-4567-24-567/08/06...$15.00

DOI: http://dx.doi.org/10.475/123_4

The idea of creating 3D contents on top of existing media is not new and often called as *modeling in context* [57, 31]. Color images remain one of the most popular visual media for such a task, since the real world can be easily acquired as a color image using a standard color camera. However, the resulting images are flat and have no associated depth information. Thus a prior step, which often requires user intervention, for recovering a certain level of 3D information from images is usually needed before such images can be used as a context for 3D modeling. We use RGB-D images, since they are easy to acquire by consumer-level depth sensors and can offer additional depth hues, largely saving us from the process of image-based 3D reconstruction.

Although RGB-D images offer us additional depth information, they do not naturally provide us a high-level understanding of the underlying image or a design space for 3D modeling. To address theses problems, we allow users to quickly express their high-level ideas via sketches, and present a sketch-based system for 3D modeling in the context of a real scene, represented as an RGB-D image (see examples in Figure 1). We focus on early stages of exploratory designs, where the creation of precise 3D models is not a requirement.

We exploit planar structures of the underlying 3D scene. However, unlike previous in-context 3D sketching systems [43, 65], which generate mainly planar canvases, our method creates a richer set of non-planar surfaces, by employing sketch-based modeling of swept surfaces (Figure 2). Hence, unlike traditional sketching systems, we let the user sketch a pair of strokes each time to create a 3D surface or surface part, which we term as a *sweep-canvas* (*s-canvas* for short).

A 2D stroke has infinite possibilities for 3D embedding. A key enabler of our system is an optimization procedure which quickly locates a 3D plane for each of the stroke pair and anchors the optimal 3D positions to sweep the strokes. To find optimal 3D planes, we first reduce the search space by assuming that each of the two strokes lies on some supporting planes which can be extracted from the existing context. To further eliminate undesired results, we enforce a set of regularizers including orthogonality, parallelism, and in-context alignments. Finally, we propose a set of candidate 3D planes to the user for confirmation and selection if needed.

Our main contribution is thus an effective sketch-based in-context modeling system, which is capable of quickly producing conceptual 3D models atop an RGB-D image for prototyping studies. Along with it, we also present a plane extraction algorithm for RGB-D images and an optimization algorithm for automatic inference of 3D strokes from pairs of 2D strokes. We evaluate our system on various RGB-D data sets captured with Microsoft Kinect and conduct a pilot user study of 10 participants with diverse backgrounds to use our system. The study shows that our system is very easy to use, feasible to novice users, and powerful in quick creation of 3D prototypes overlaid with existing context.

RELATED WORK

Our work is closely related to the researches on sketch-based modeling and interpretation. Below we review the most relevant works in the area of sketch interpretation, 3D sketching, in-context modeling, and 3D modeling with RGB-D.

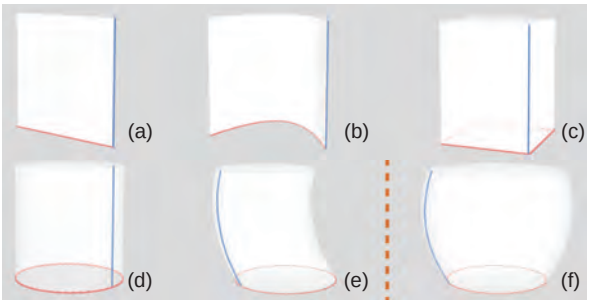


Figure 2. Basic swept surfaces created by pairs of profile strokes (red) and trajectory strokes (blue). (f) is derived using the same strokes in (e) with symmetry enforced.

Sketch interpretation. In the past decades, a heavy body of research works have been devoted in the areas of sketch-based shape retrieval [14, 61] and interpretation [10, 48, 49]. Sketches have also been extensively exploited for 3D modeling due to its nature of flexibility. The famous Teddy system [23] is perhaps one of the pioneer works in sketch-based freeform 3D modeling, with the follow-up works including [53, 40, 37, 59], whose target is to interactively create freeform 3D shapes by properly interpreting user sketches. Instead of creating 3D shape on the fly, various sketch-based modeling systems take as input a complete sketch and analytically analyze individual strokes [33, 7, 41, 60], geometric relations among strokes [18, 54], or network of 3D curves [45] to extract 3D models. Please refer to the work of [39] for an excellent survey on sketch-based modeling. Our work is different from the existing sketch-based 3D modeling techniques at least in two aspects. First, instead of creating watertight models or parts, we intend to create and assemble a collection of swept surfaces for 3D prototyping, in a spirit of the Paper3D system [42], which provides a multi-touch interface for assembling complex 3D scenes from a collection of developable surfaces. Second, our approach heavily exploits in-context information to help create and position 3D surfaces.

3D sketching. Limited by the input devices or media, most sketching and modeling systems are essentially 2D [44]. Thus, in parallel to the set of works on sketch-based modeling, there is a line of researches which focus on directly mapping 2D sketches into 3D. Studies on lifting 2D sketches into 3D have been explored in the systems such as Mental Canvas [13], iLoveSketch [1], and EverybodyLoveSketch [2]. These systems exploit pre-anchored 3D planes to lift the 2D strokes into 3D. Schmidt et al. [46] use scaffold lines for analytical drawing in 3D. A recent work of *SketchingWithHand* anchors 3D planes by human hand for sketching hand-holdable objects [29]. The *TiltBrush* [19] by Google offers a substantially different input by allowing users to directly draw in the 3D space using the 3D tracking devices. In our work, the back-projection of 2D strokes to 3D is an intermediate step. The projected strokes will later serve as profile and trajectory curves to create our final 3D surfaces. Nevertheless, in the decoration period, the user can draw sketches over the created

surfaces to author 3D sketches on top. Hence, 3D sketching can be a by-product of our pipeline.

Modeling in context. A design process typically involves conducting studies on prior art. Sketching on top of existing visual contents could largely inspire such design processes [27]. 3D sketching systems such as 3D6B [28], Sketching Reality [7], Insitu [43], SecondSkin [11], and SmartCanvas [65] are created in the line of such workflows. Favreau et al. [16] exploit estimated orientations from an underlying point cloud to reconstruct line drawings. Lau et al. [31] further physically fabricate the modeled object on a single photograph. Gannon et al. [17] enable a direct modeling and fabrication work flow leveraging human skins, and Huo et al. [21] use Google Tango to create simple 3D shapes on top of a video by performing sketch-and-inflate operations, mimicking a mixed-reality experience. A recent work of 3-sweep [6] allows users to interactively create swept surfaces on top of an RGB image. However, since their focus is on 3D reconstruction of image objects, their method cannot create objects which do not exist in the input image. Unlike previous systems such as SecondSkin [11], SmartCanvas [65], and Insitu [43], which require as input either a complete 3D model, a set of pre-defined 3D planes or computed point cloud, our input is a single unstructured RGB-D image, which is lightweighted, realistic, and easier to acquire.

Modeling with RGB-D. As RGB-D images offer us additional information about depth, they have been widely exploited for various tasks like 3D reconstruction [25, 52, 51, 63], object detection [56], and image understanding [50]. Our method creates 3D models on top of an RGB-D image. However, as a key difference, our focus is not on object or scene reconstruction but creating new models which do not exist in the image by exploiting contextual information. Hence, our setting is more tightly tied to applications like augmented reality. A recent work of [38] introduces a method for real-time snapping of 3D models into real scenes captured by depth cameras. They also exploit the underlying linear features of RGB-D images. However, the framework settings and the main goals are very different between their method and ours. We exploit planar structures to lift 2D strokes into 3D while their focus is on the extraction of edge and plane constraints for 3D snapping.

BASIC NOTIONS

We begin with the introduction of a few basic notions. The basic 3D elements that constitute our system are *swept surfaces*, each created from a pair of user strokes. Like in any conceptual sketching systems, we allow further decoration and painting operations on the generated surfaces. Hence we term such a basic swept surface as a *sweep-canvas*, and *s-canvas* for short.

In mathematics, given two 3D curves $c_1(u)$ and $c_2(v)$, a *swept surface* is the surface generated by moving curve $c_1(u)$ along curve $c_2(v)$. Curve $c_1(u)$ may be rotated and scaled. More precisely, for each v in the domain of curve $c_2(v)$, curve $c_1(u)$ is moved to the point $c_2(v)$, possibly with rotation and scaling. Therefore, as v changes from 0 to 1, the transformed curve $c_1(u)$ sweeps out a surface and hence the name swept surface. Under this definition, curves $c_1(u)$ and $c_2(v)$ are referred to as the *profile curve* and *trajectory curve*, respectively. In modeling systems such as AutoCAD, SKETCH [62], SketchUp [58] and Pushpull++ [32], swept surfaces are created by sweeping a base 3D profile curve along a 3D path (trajectory) curve, whereas the path curve is normally a predefined straight line. In contrast, we take 2D curve strokes as input and infer their 3D positions using the context information from a given RGB-D image. Figure 2 illustrates a few basic swept surfaces supported by our system. In our implementation, rotation but not scaling is allowed for the profile curve. To the best of our knowledge, it is for the first time that the creation of such swept surfaces using casual sketches on top of an RGB-D image is introduced.

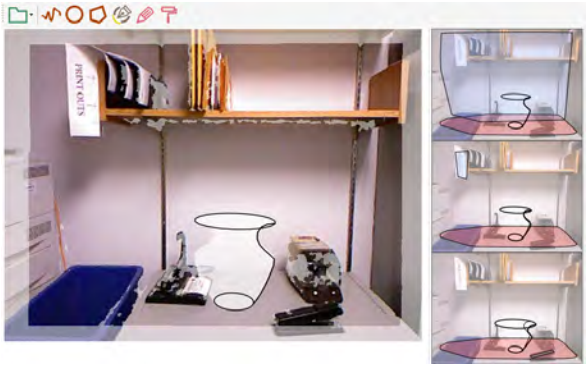


Figure 3. Our user interface consists of two main panels: a sketching panel where the user sketches on top of an RGB-D image to create s-canvases and a candidate panel where best-ranked candidate s-canvases are listed. The red and blue polygons are the supporting planes for the profile and trajectory strokes, respectively.

USER INTERFACE

Given these notions, we now unfold the design of our system. Our user interface (Figure 3) contains two panels: a main window for 3D modeling via sketching over an RGB-D image and a candidates selection panel constituted of a set of sub-windows.

Sketching. By default the user enters the sketching mode where s/he can sketch over a pre-captured RGB-D image using either a mouse or a touch pen. Our system takes as input a pair of strokes, namely a *profile* stroke and a *trajectory* stroke. We assume both strokes lie on some unknown spatial planes, and the first stroke is a profile stroke, followed by the second one as a trajectory stroke. During the user draws the trajectory stroke, our system automatically creates an s-canvas and returns it to the user. One or multiple candidate s-canvases will show up in the sub-windows on the right panel for the user's selection in case the desired one is not in the main sketching window. Drawing multiple profile strokes ie enabled by pressing the 'ctrl' key. Although the suggestive interface has been seen in the previous works like *Chateau* [22], *Chateau* does not exploit contextual information nor could it create swept surfaces aimed for AR applications.

To create a new s-canvas, the user can repeat the above process, or first select an existing stroke or its part as a profile stroke and then draw a new trajectory stroke (Figure 4 (a) and (b)). S/he can also make use of two existing strokes (Figure

4 (c)), or draw the profile stroke directly on top of an existing canvas, followed by a trajectory stroke (Figure 4 (d)). Creating new canvases by reusing existing strokes provides a smooth modeling process since a new canvas typically shares relations with the existing ones.

Viewing. The user can change the view of the scene to get a clear sense of the created 3D contents and continue to draw in that view (Figure 6). Change of views is enabled with rotation, zooming, and panning using a track-ball implementation. To seamlessly integrate with sketching and avoid tedious mode switching, the rotation is continuously enabled with the right mouse button across different modes while the left mouse button is mainly used for sketching, editing, painting, etc. In cases of a stylus or a touch pen, the features related to the right mouse button are enabled by pressing a button either directly on the device or on a keyboard.

Auxiliary operations. Our system allows a set of auxiliary operations to enrich and smooth the modeling process, namely, editing, painting & coloring, and plane manipulation.

In the editing mode, the user can edit the shape of an existing s-canvas in two ways. First, the user can re-sketch either the profile or the trajectory strokes of the s-canvas (Figure 5 (a)). Second, the user can cut the s-canvas by sketching. For example, the user can sketch a long stroke to cut a part of an s-canvas or draw a closed stroke to cut out a hole from the canvas (Figure 5 (b)). The user can also erase using a painting interface or delete an s-canvas (see in the accompanying video). Additionally, copy and paste operations are supported with keyboard shortcuts as 'ctrl + c' and 'ctrl+v'.

The user can enter the decoration mode to either color an s-canvas or draw decoration strokes on the s-canvas to add details, like in a 2D illustration system. Figure 5 (c) shows an example of painting. All the user-drawn strokes are projected onto the 3D s-canvases by ray-casting. The user can also erase or delete any painted strokes. In the coloring submode, the user can color the s-canvas and decoration strokes using a standard color palette interface.

To increase the flexibility of our framework, our system allows the user to manually specify and manipulate a 3D plane to serve as supporting plane for the profile or trajectory stroke. More specifically, the user can click on the image to select an existing plane (those extracted) and then translate or rotate

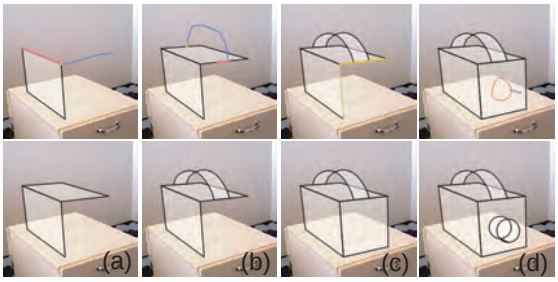


Figure 4. Various cases to create new s-canvases: (a) and (b) reuse an existing (red) profile stroke or its part; (c) select two existing strokes (yellow); (d) draw a profile stroke on an existing canvas and draw a new trajectory stroke.



Figure 5. Auxiliary operations: (a) edit a profile or trajectory stroke to edit an existing s-canvas; (b) cutting; (c) painting and coloring.

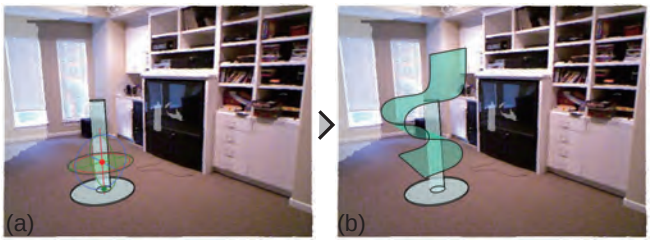


Figure 6. Plane manipulation operation: an existing plane in the context can be transformed in space (a) for creating a new s-canvas (b).

the plane to a desired position for stroke drawing. See Figure 6 for an example.

ALGORITHM

The input to our system is a single RGB-D image captured by a consumer-level depth sensor, Microsoft Kinect V1 in our case. Alternative RGB + depth sensors can also be used. We first perform structure analysis on the underlying scene to extract a set of dominant planes from the RGB-D data. These planes will serve as reference planes for the user strokes. Then, a set of image features such as corner points and edges are consolidated with the depth data to get a set of localized 3D features. We use these features to help position and create the s-canvases. Then in a key stage, our algorithm selects a candidate plane for each of the profile and trajectory strokes via an MRF (Markov Random Field) labeling process, and optimizes their positions by analyzing stroke-to-context relations.

Plane extraction

Our algorithm assumes that the user-drawn strokes lie on some spatial planes. To find these planes, we first analyze the RGB-D data to get a set of candidate planes. Each pixel $u(x, y)$ of the RGB-D image comes with a color rgb and a depth value d. A straightforward method to get a set of planes from the depth data is to run RANSAC [47] directly on the input. We find that such methods do not always return clean results (see Figure 10 (b)). Although alternative sophisticated plane extraction algorithms might be applied here (e.g., [24, 35]), we find the following simple strategy works very well throughout our experiments.

Our goal is to identify the main planar regions from the RGB-D data. Hence we first compute the normal for each pixel data. A direct method to compute normal is to simply use the neighboring pixels (left, right, up, bottom) and compute their vector cross product with respect to $u(x, y)$ [66]. This, however, is erroneous since the captured data using Kinect V1 is usually very noisy. We resort to a method similar to [20]. Specifically, given each point, we find all points in a sphere with radii r ($r = 7$ pixels in our implementation) and

fit these points with a best plane using RANSAC. We use the plane normal as the normal for that point. As a consequence, points lying on planar regions get consistent normals while points lying on the boundaries might get incorrect estimations, which are thus ignored in our later clustering process (Figure 7 (c)).

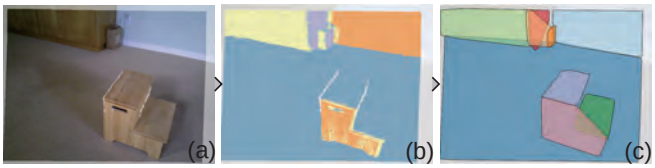


Figure 7. 3D plane extraction. (a) Input RGB-D data. (b) Clustered patches. (c) Extracted planes.

We run meanshift [8] on the RGB-D data coupled with the computed normal, where each data point is now denoted as $u(x, y) = (r, g, b, d, n_x, n_y, n_z)$. This gives us a set of patches, where the points with similar colors and normals are clustered (Figure 7 (c)). We then run RANSAC on the generated patches of these regions to fit planes one by one. Once a plane is extracted we remove the associated points and run the RANSAC on the rest. The iteration stops when a newly fitted plane does not contain a sufficient number of points (5% of the points in our experiments). This process returns a cleaner set of planes, compared to a direct RANSAC as shown in Figure 10.

3D feature consolidation

Sketching new contents atop existing context amounts to exploiting in-context relations. For example, when the user draws a side holder along the chair seat or an arm attaching to the chair seat and chair back, the drawing strokes should be aligned or attached to the 3D features such as edges and corners of the chair (see Figure 9). Thus we should extract the key features from the RGB-D data to help the positioning of 2D strokes in 3D. We introduce the term *attachments* to accommodate such stroke-to-context alignment and find the following three types of attachments are particularly useful to interpret the user's intention: attatch to an existing planar face, a 3D edge or a 3D corner point.

As we already have planes in hand, we only need to detect the 3D edges and 3D corner points from the RGB-D data. Edge detection and corner detection are two fundamental problems in computer vision and have been heavily studied in 2D [12].

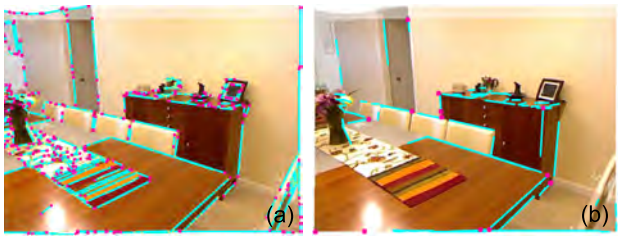


Figure 8. 3D feature consolidation. Left: 2D image features (edges and corners). Right: 3D features extracted by consolidating the 2D features with the depth data. We also include the intersecting line segments and corners of the existing planes.

However, relatively few of research works have focused on the case of an RGB-D image. A direct detection of these features in the RGB space will disregard any captured depth information and lead to a redundant set of features (e.g., the texture edges in Figure 8 (Left)). An intuition on extracting edges from depth data is to first compute 3D planes and the edges are naturally their intersections [24, 4]. In the following, we design a viable solution particularly tailored for our system, which bears the similar idea of model fitting [24] for edge estimation using depth.

We first detect the edge and corner features in 2D. We then consolidate these features in 3D to yield a set of 3D edges and corners. In particular, we examine the 2D edges and corner points detected using standard vision techniques [3, 12] with the extracted 3D planes. As shown in Figure 7, a plane in our case is bounded by the convex hull of its associated points. For a 2D edge, we consider it as a 3D edge if it agrees with one of the boundary line segments of any plane in Ω. We examine the proximity in both direction and position between the 2D edge and the plane boundary line segments. Similarly, for a 2D corner point, we tag it as a 3D corner point if it lies in a local window (size of 20 pixels) of any 3D corner points of the detected planes in Ω. The filtering process leads us to a much cleaner set of 3D features. Figure 8 (Right) shows an example of detected 3D edges and corner points in the dining room example. Although the detected 3D corners and edges might still contain noisy estimation, especially in image parts which have inaccurate depth data due to their large distance away from the camera, we find it to be sufficiently clean and helpful in our subsequent canvas creation algorithm.

s-Canvas creation

We now detail our canvas creation algorithm. As mentioned before, we assume that the profile stroke and the trajectory stroke both lie on some unknown spatial planes. These spatial planes can be the existing planes in the context or the planes that are parallel to the existing planes. The parallelism assumption boosts the creation of s-canvases not limited to the existing planes as such restriction may significantly limit the space of s-canvas creation.

Note that the parallelism assumption has been exploited in the work of [65], where they also assume the user-drawn strokes lie on some planes parallel to existing planes. However, as a key difference, we do not exhaustively sample a dense set of parallel planes pre-hand to look for the best ones. In contrast, we decouple the process by separating the plane orientation estimation from the plane position estimation, i.e., we first determine plane normals and then plane positions. Such decoupling not only reduces the search space (thus enabling us a full exploration of the search space), but also avoids the post-optimization process required to precisely locate the planes with respect to the context as required in [65]. We show next how this strategy benefits our canvas generation algorithm.

Plane orientation selection. Let us denote the set of planes extracted from the previous section as $\Omega = \{p_1, p_2, ..., p_n\}$. Our task is to select for the profile stroke s_p and the trajectory stroke s_t supporting planes p_i and p_j, respectively, such that the strokes s_p and s_t lie on some planes that are parallel to p_i

and p_j, respectively. A simple local strategy could easily lead to undesired results as the strokes are frequently overlapping with multiple planes (e.g., the ground plane and the walls, see in supplemental material for an example). To overcome this, we free the space to explore a larger set of possible planes while coupling the strokes with the underlying depth data. Essentially, this results in an MRF labeling problem, which can be formulated as the following minimization problem:

$$\begin{aligned}\underset{\{i,j\}}{\operatorname{argmin}} E_{i,j} :=& \{\lambda_p E_U(s_p \to p_i) + \lambda_t E_U(s_t \to p_j) \\ &+ \lambda_s E_B(s_p \to p_i, s_t \to p_j)\},\end{aligned} \tag{1}$$

with $i, j \in \{1, 2, ..., n\}$. We define the unary and binary terms to score the selection.

The unary term is based on the observation that when in action straight lines and in-plane strokes are usually preferred by the user. In practice, we measure the following three factors: how likely a drawn stroke lies inside a plane and how likely a drawn stroke approximates a straight line. The unary term, in general cases, is defined as:

$$E_U(s \to p) = \psi(s \mid p) \otimes \eta(s), \tag{2}$$

where $\psi(s \mid p)$ is the likelihood of a stroke s lying on the plane p, defined as $e^{-\tau(s|p)^2/2\sigma_p^2}$ with $\tau(s \mid p)$ denoting the portion of the stroke points which are inside the plane p (for a closed stroke, we measure the overlap region (in terms of area) between the stroke and the plane in 2D). σ_p is set to 0.45. $\eta(s)$ is a delta function whose value depends on both s_p and s_t. In specific, it takes the value of 0.5 if both strokes approximate a straight line (to favor in-plane straight lines) and 1 otherwise.

The binary term is defined to avoid selecting nearly parallel planes, which will cause undesired sweeping results, while on the other hand to encourage the selection of planes which are approximately orthogonal. We define it as follows:

$$E_B(s_p \to p_i, s_t \to p_j) = e^{-\theta_{i,j}^2/2\sigma_t^2}, \tag{3}$$

where $\theta_{i,j}$ is the angle (with the range from 0 to $\pi/2$) between planes p_i and p_j, and σ_t is set to $\pi/4$ to enforce the binary term to favor a pair of planes which are orthogonal to each other.

In our experiments, we set $\lambda_p = 3, \lambda_t = 1$ and $\lambda_s = 2$, to emphasize the fidelity of the profile stroke drawn by the user. This is based on the observation that a profile stroke typically will have more direct relations to the existing context than a trajectory stroke (recall our assumption that the profile stroke shall lie on some existing plane.) To avoid cases when there is no desired plane in the context, we add the following case in our implementation: if a user stroke s approximate a straight line (examined in 3D), we add a plane which is orthogonal to s as a candidate plane.

The above labeling problem can be effectively solved via graph cut [30]. We observe that the variable space is small (2 in our case, i.e., the two strokes). Hence a complete enumeration only leads to a time complexity of $O(n^2)$, with n the number of extracted planes.

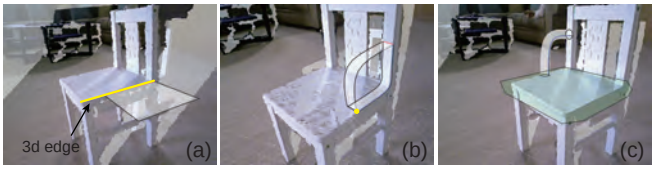


Figure 9. Various snapping cases enabled in our system: (a) snap to edge, (b) snap to point, and (c) snap to face.

Plane localization. The above process gives us a pair of planes (one profile plane and one trajectory plane, c.f. the red and blue planes in Figure 3), which exist in the context. The actual positions of these two planes can deviate by some offsets along the normal directions (recall our the parallelism assumption). Our next task is thus to locate the spatial positions of the two planes by finding the correct offsets. The key here is to derive some useful cues from the user-drawn strokes to localize the plane positions. To this end, we analyze the stroke attachments to the existing context.

We find that the following attachment analysis is essential in locating the final planes. First, the intersection point between the profile stroke and the trajectory stroke often indicates the start position of the sweeping. Second, the start and end points of the strokes are often attaching to some planes, edges, or corner points in the image when the user draws in the context.

Essentially, our goal is to find some *attaching point* of the strokes to the RGB-D data that the profile and trajectory planes should pass through. Let us denote the set of 3D features extracted in the previous process as F (Section 5.2). Our analysis operates subsequently, and examines the following types of attachments in order, to locate the points. The first is to examine if the intersection point between the profile and trajectory strokes coincides with some corner point or lies on some edge. The intersection point is found as the nearest point of the profile stroke to the trajectory stroke. We search in a local window of 20 pixels of the intersection point in the RGB-D data and set the corner point with the least depth w.r.t. the current viewing direction in F as the 3D location for both the profile and trajectory planes. If such a corner point is not found, we continue to search for the attachment of the two strokes with the existing 3D edges. If any stroke is aligned or partially aligned with an existing 3D edge in F, we set the projected point on the 3D edge as the attachment for the two planes. Finally, if none of the above cases is found, we set the attachment as the most front point in the local window w.r.t. the current viewing direction.

Candidates ranking. The selection process results in a set of pairs of planes with their selection costs computed according to Equation 1. We rank them according to their scores and push them into the candidate panel for the user to select in case the desired result is not in the main window. We filter out plane pairs which will lead to similar s-canvases, by eliminating all plane pairs which have similar orientations to the previously selected plane pairs. Figure 3 shows an example of pushed canvases in the candidate views.

In-context snapping. The above process gives us a swept surface created from two planar strokes. To facilitate more

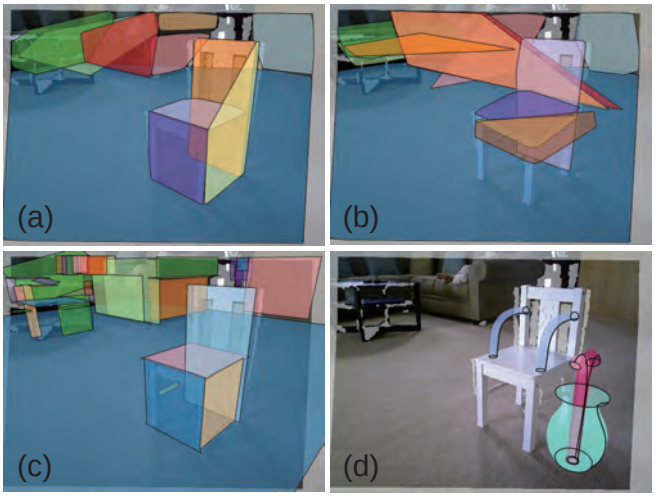


Figure 10. Evaluation of two plane extraction methods. (a) Planes extracted with our method. Because of the visualization, the yellow plane at the right side of the chair is correct but looks not, similar for the rest of the planes far away. (b) Planes extracted using RANSAC. (c) Planes extracted with the method by Monszpart et al. [35]. (d) The target example to draw.

fluent drawing experience, our system supports stroke snapping to existing context in the following three cases. Firstly, if the current profile stroke moves close to an existing canvas or 3D plane, we inform the user possible snapping by highlighting the plane (in green in Figure 9 (c)). Secondly, if the 3D position of the current sweeping stroke end point is in close proximity to an existing 3D corner point or 3D edge (both in the RGB-D context or in the created s-canvases), we highlight the corresponding point (in yellow in Figure 9 (b)) for snapping. Please see various cases of snapping in action in the accompanying video.

EXPERIMENTS

We tested our framework on various scenes captured by Kinect. We focus on man-made scenes where planar structures are ubiquitous.

Exemplar scenes. The scenes we collected include examples like a computer room in a university, desktop workplace, street halls, coffee room, kitchen, laboratory, etc. (Figure 11). They span a typical set of scenes in man-made environments. Most of the scene data came from online resources, including but not limited to NYU depth v2 [36], Berkeley B3DO [26], and RGB-D datasets from [9]. We also captured some using our Kinect V1 device.

Statistics. Figure 11 shows examples created using our system. In general, our system is capable of creating models with regular or isotropically curved shapes such as furniture, toys, and other shapes in man-made environments, which are constituted of primitive-shaped parts (see in the supplemental material for more results). Our system currently does not support shapes with anisotropic scaled parts, e.g., a shell.

Table 1 indicates the numbers of canvases and the total time for the examples in Figure 11. On average it took an approximate time of $5 \sim 20$ minutes to create each example, including the idling time. All examples were created on a Laptop

Stat.	truck	Chn style	piano	kitchen	console	hallway
#canvas	21	26	31	48	48	73
time (min)	6	8	15	15	29	32

Table 1. Statistics recorded for the examples in Figure 11.

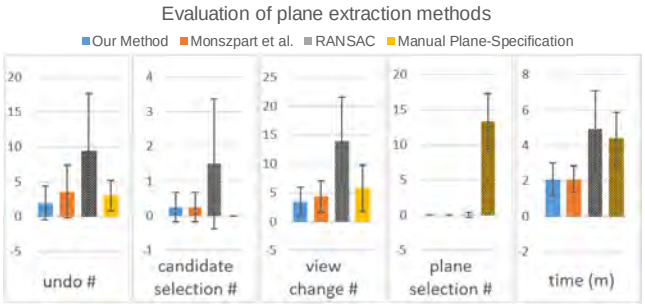


Figure 12. Comparison statistics of our plane extraction method against RANSAC, the method by Monszpart et al. [35], and manual plane-specification. Planes and example scene are shown in Figure 10. The error bars show the standard errors of the mean (SEM) of the numbers among different users. Bars with hatching lines indicate the results of Dunnett test where there was a significant difference between the specified technique and our technique in the given criteria.

with an Intel(R) Core(TM) i7-2620M 2.7GHz CPU and 8GB RAM.

Evaluation. We evaluated various algorithmic components of our system. We first compared our plane extraction algorithm with the standard RANSAC method [47] and the method by Monszpart et al. [35]. Note that we are not claiming a better general method towards plan extraction. So we focused our evaluation on how the modeling performance would change if we use the planes from the alternative methods. We asked 8 users (3 with 3D modeling experience and 5 with no modeling experience) to experiment with the chair scene example shown in Figure 10. They were asked to draw the same example using the planes extracted with (1) our method, (2) RANSAC [47], (3) the method of [35], and (4) manual selection. Note the manual selection process mimics the previous baseline modeling interfaces as in Google SketchUp [58], AutoCAD, and SKETCH [62], which use manual or predefined planes to anchor swept surfaces. We implemented by allowing the user to manually select two planes for both profile and trajectory strokes each time to create a shape part. We disable any automatic inferring and snapping features in this mode. The user has to manually translate and rotate the plane to a desired position and then draw a profile or trajectory stroke to sweep a surface.

We ran a Latin square test with the 4 methods and 8 people and recorded the operation statistics for both settings and all users. These statistics included the total number of created s-canvases, the numbers of various standard operations performed (e.g., view changes, undo, candidate selection, plane manipulation, canvas transformation, edits, etc.), and the total time used to create each result. Figure 12 shows the comparison results with the numbers averaged over the 8 users. The bars with hatched lines show statistically significant results of a Dunnett test. The tests were conducted as multiple comparisons, in which our method was compared with all other three techniques.



Figure 11. Various 3D conceptual models created using our system. Our system is capable of quickly anchoring desired s-canvases on top of the given contexts. Most of the examples were sketched within 20 minutes.

It is not surprising that compared to our method RANSAC needed a much longer time and more undo operations to create the desired results due to the redundant and undesirable set of planes extracted, resulting in frequent incorrect inferences. Similarly a longer time was recorded for the manual plane specification (c.f. a baseline approach), since the users had to position the planes first, which is a time-consuming process. It is interesting to note that the time for manual plane selection is even less than that for RANSAC, possibly because the erroneous planes in RANSAC frequently led to undesired surfaces (e.g., the chair arm) and the users thus had to rotate views and redraw several times in order to get a reasonable result. Still the change of views is dependent on individual perspectives, resulting in relatively large variance among the users.

The statistics for ours and the method of Monszpart et al. [35] were similar given that the generated planes were both similar and clean. Indeed, at regions with large noise where the depth sensor got very unstable estimates, our method might not perform as good as Monszpart et al. [35] (see Figure 10), since their method implicitly enforces a set of regularities in the plane arrangements (e.g., angles, symmetries) while our method does not. Yet, such regulations might lead to errors (e.g., the planes generated around the leg part of the black table are not correct in Figure 10). Nevertheless, our method is much simpler and ran a magnitude of 5 times faster. We also conducted a set of experiments comparing the three algorithms with the ground truth planes obtained manually (Figure 13). The error measures agree with our visual findings (Table 2).

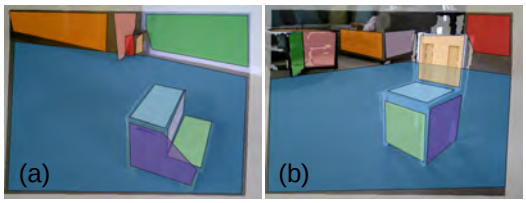


Figure 13. Ground truth planes for scenes in Figure 7 and Figure 10, respectively.

Scene	Method	Mean (SD)	Precision	Recall
Stair	RANSAC	4.01°(**2.10**°)	14.28%	66.67%
	Monszpart	3.16°(3.64°)	44.44%	91.67%
	Ours	**2.87**° (3.77°)	**50.00%**	**92.39%**
Chair	RANSAC	4.73°(5.40°)	33.33%	67.18%
	Monszpart	**2.00**°(**2.39**°)	25.00%	**90.62%**
	Ours	2.61°(3.01°)	**62.50%**	90.60%

Table 2. We compare the relative angles between the normals of planes extracted from different algorithms and ground truth planes. "Mean (SD)" means the mean and standard deviation of absolute difference of relative angles. We were interested in how relative planes were successfully extracted perfectly, using 1° as a reference. Many points were sampled in those extracted planes. A point can be known as recalled, if it is close to its relative ground truth plane (distance $< 5e-4$). The percentage of the method of Monszpart et al. in column 4 is lower because their method fitted many false positive planes around the sofa. Many points were sampled in those extracted planes.

PILOT STUDY

To further evaluate our system, we conducted a pilot study on a small group of 10 participants, whose background were quite diverse. Among them, 3 people (called as experienced users for short) were experienced users in 3D modeling, 4 people (i.e., beginners) had preliminary experience in 3D modeling and sketching, and 3 people (novice users) had no drawing or modeling experience at all. Most of them were graduate students from universities, who were majored in computer science, social media, and digital art. The participants were provided with professional Wacom graphic tablets (without display panels) and traditional mouse devices. We did not enforce a stylus input since many of our participants were not familiar with a stylus input. Two participants who had previous experience in sketching chose to use the tablet, while the others mainly used a mouse for sketching. We found our system is sufficiently robust to tolerate imprecise input from a mouse. This is mainly because our system implicitly smooths the user sketches (see in our accompanying video).

Training. For all the participants, we first briefed them and trained their skills on using our system. We showed them a video demonstrating all instructions on how to create various basic s-canvas, how to create a new s-canvas by reusing existing s-canvases, and how to use auxiliary operations such as editing, painting, plane manipulation, etc.. After the tutorial, they had a 10-minute free practicing time, during which they were free to try out our software and ask questions.

Next, we designed three particular tasks for them to explore.

Task i. In the first task, we asked them to draw two simple examples, as shown in Figure 14 (a) and (b).

Task ii. In the second task, we asked them to draw two relatively more complex examples, as shown in Figure 14 (c) and (d). In this task, both the model complexity and the auxiliary operations/contextual information required for creating the model increase.

Task iii. In the third task, each participant was invited to draw free examples on top of a given scene. A total of 10 scenes were given to them in this task.

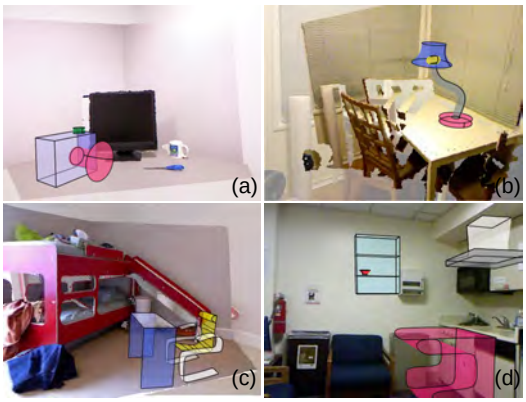


Figure 14. Exemplar reference models used in Task i (a), (b) and Task ii (c), (d) of our pilot study.

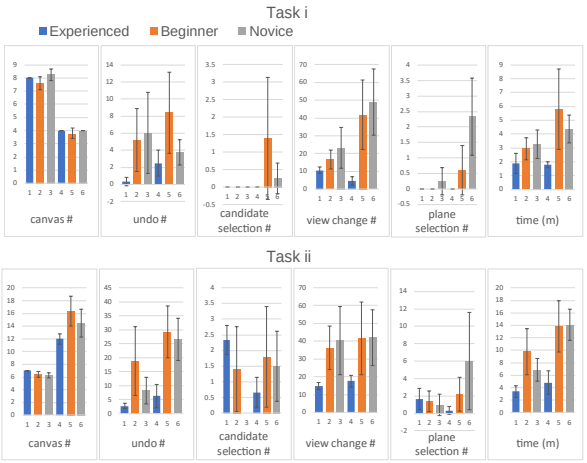


Figure 15. Statistics for the pilot study of Task i and Task ii. Each task contains two examples, corresponding to the four examples in Figure 14. The error bars shows the SEM of the numbers among different users. Note novice users could learn our tool fairly fast compared to experienced 3D modeling users.

Statistics. Figure 15 shows the statistics of the averaged numbers of operations and time for Tasks i and ii. The six bins on each operation contains the statistics for the two examples in each task (3 bins per example). It can be observed that, as the example complexity increased, the time required to mimic each example increased, roughly linearly. Again, all examples took a rough time of $5 \sim 15$ minutes to finish, even for novice users who had no experience in 3D modeling at all. More importantly, the candidate selection operation kept at a small ratio throughout all the experiments, indicating the robustness of our system. We also observe that the auxiliary "plane manipulation" operation was rarely used and performed mainly for creating floating surfaces such as the ones shown in Figures 6 and 14 (c). Some non-experienced users utilized this operation for creating the smoke extractor in Figure 14 (d). The undo operation was more frequently used by non-experienced users and the beginners, since their drawings were in many cases imprecise but they tended to have more willing to draw well, thus resulting in such higher numbers of the undo operation.

We also designed a standard user study questionnaire for them to fill out once they had finished the study. We found that most of the participants were able to quickly learn how to use our system, and found our system convenient to use and the interface simple and clean. 6 of the users who had previous experience in 3D modeling thought that our system offered a very intuitive and creative way to create 3D models on top of an image, perhaps one among the simplest. They commented that the candidate selection interface could somehow disturb the modeling process as it required switching back and forth. Nevertheless, the user study shows that this process was kept at a low ratio during the modeling process. Please refer to the supplemental material for more examples drawn by these participants.

DISCUSSION AND LIMITATIONS

Context utilization. Our approach largely leverages the context to anchor the drawn strokes. The context being used is

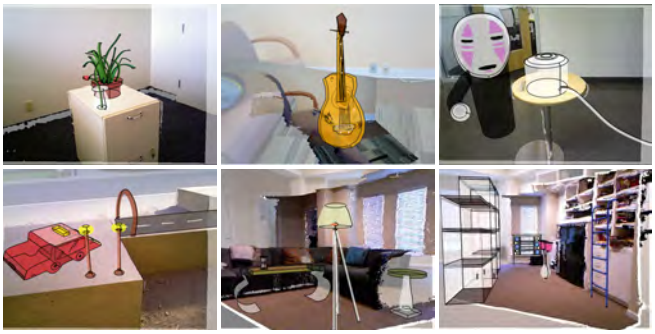


Figure 16. Representative models created by our participants in Task iii.

mainly the identified 3D attachments. For example, when creating a bookcase (Figure 1), chair arm (Figure 9), gas filter (Figure 14), or ladder (Figure 16), frequently, the strokes are snapped to the 3D planes, edges/corners as well as the depth data itself (for local positioning). Once some parts are anchored, the rest of the parts can be created by utilizing the already created parts for better positioning. The extent to which the existing context is used largely depends on how many relations the created parts share with the context. We observe that the users with professional design skills tended to draw parts that bear more relations to the existing context, such as ladders, bridges, wall-to-wall tubes, etc., while non-skilled users tended to draw simpler cases such as lamps, bookcases, etc. and paid less attention to the context.

Extension to photographs and videos. Our pipeline uses RGB-D images as our main media for contextual inference and the subsequent modeling. It can be naturally adapted to other simpler types of media such as a single photograph and a video if augmented with depth information. In a single photograph, there are multiple ways to acquire the necessary 3D information for instance using a depth inference algorithm [34] or a set of extracted boxes [64]. For a video, the state-of-the-art SFM and SLAM techniques [55, 15] can be utilized to construct a dense point cloud from the sequential images. Please see our accompanying video for such an example of modeling-in-video. 3D models or environments [62] is another choice but might exclude the user from an experience of modeling in "reality". Alternatively, one could also leverage a simple box case as in [65] as a start, however, such simple initialization could depart the user from utilizing the context information thus limit the creation capacity.

Expressiveness. Our tool mainly aims for quick and easy 3D prototyping. In this work, we have focused its usage on modeling furniture-level objects in indoor scenes due to the acquisition capability of consumer-level depth sensors. While the resulting models are relatively low-resolution, the high expressiveness of our tool can be easily seen from the examples and it is thus useful for concept design.

More detailed modeling features such as profile/trajectory stroke editing, surface sculpting, extrusion, sharping, could be easily integrated into our tool set. Alternatively we may enrich details by retrieval-based approaches (e.g., [5]). Yet, the more challenging thing is to create arbitrary spatial curved surfaces such as tree branches, for which we hope to explore more in the future by utilizing the contextual information.

Limitations. Our system requires two user input strokes to create a 3D s-canvas. In general cases, the drawn strokes lead to desired results. However, there are occasions when the two strokes fail to span a desired s-canvas. This happens when the direction of drawn strokes is almost parallel to the viewing direction, leading to effects such as foreshortening. Our system will omit such long projection as it is undesirable. Additionally, failure cases are also possible when a desired 3D supporting plane does not exist in the context nor can it be added by our algorithm (e.g., a floating cuboid). In such cases, the user needs to perform the auxiliary operation of "plane manipulation" to manually create such a 3D plane. Our system currently only exploits planar structure of the underlying scenes. More complex relationship such as repetition, spacing, or curved surfaces are not considered. Although they can be extended in general, however, extracting such information from low-accuracy depth data could be an overhead. We leave this for future work. Finally, our system currently does not support the scaling of the profile stroke hence it can not create shapes such as a tobacco pipe. Advanced editing interfaces such as those profile editing tools in conventional CAD systems could be implemented to accomplish such tasks. However, for simplicity and compactness, we do not include them in our system.

CONCLUSION AND FUTURE WORK

In conclusion, we have presented a sketch-based modeling tool for novice users, aimed at fast 3D prototyping on top of images or videos (RGB-D images in our case). We formulate the modeling task as creating swept surfaces (s-canvases) from two 3D strokes drawn by the user, where the strokes are embedded into a pair of 3D planes. Our tool is intuitive, easy-to-use for end users, and effective in creating a diverse set of surfaces in man-made environments. Our system largely exploits the RGB-D information in the context and leverages planar structure analysis to help anchor the user drawn strokes into 3D to create swept surfaces. The core of our formulation is an MRF-based optimization procedure to quickly locate 3D planes from the context for the two strokes. Our method supports interactive conceptual modeling by instantly optimize the created 3D model on-the-fly and returns it to the user in accordance with the drawing strokes, enforced with a set of shape regulators such as in-context snapping, orthogonal enforcement, symmetry etc. In the future, we plan to experiment with more different media such as single images or videos. On the other hand, we believe that such conceptual modeling tools can be naturally adapted to other applications in VR and AR, which is our future work.

ACKNOWLEDGEMENTS

We thank all reviewers for their valuable comments. This work was supported in part by the National Natural Science Foundation of China NO. 61502306, NO. 61602310, the China Young 1000 Talents Program, Shenzhen Innovation Program NO. JCYJ20170302154106666, and the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU11300615, CityU11204014, CityU113513).

REFERENCES

1. Bae, S.-H., Balakrishnan, R., and Singh, K. Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models. In *UIST '08* (2008), 151–160.
2. Bae, S.-H., Balakrishnan, R., and Singh, K. Everybodylovessketch: 3d sketching for a broader audience. In *UIST '09* (2009), 59–68.
3. Canny, J. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, 6 (1986), 679–698.
4. Casarrubias-Vargas, H., Petrilli-Barceló, A., and Bayro-Corrochano, E. Fast edge detection in rgb-d images. In *Iberoamerican Congress on Pattern Recognition*, Springer (2014), 868–875.
5. Chaudhuri, S., and Koltun, V. Data-driven suggestions for creativity support in 3d modeling. *ACM Transactions on Graphics (TOG) 29*, 6 (2010), 183.
6. Chen, T., Zhu, Z., Shamir, A., Hu, S.-M., and Cohen-Or, D. 3-sweep: Extracting editable objects from a single photo. *ACM Trans. Graph. 32*, 6 (Nov. 2013), 195:1–195:10.
7. Chen, X., Kang, S. B., Xu, Y.-Q., Dorsey, J., and Shum, H.-Y. Sketching reality: Realistic interpretation of architectural designs. *ACM Trans. Graph. 27*, 2 (May 2008), 11:1–11:15.
8. Cheng, Y. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence 17*, 8 (1995), 790–799.
9. Choi, S., Zhou, Q.-Y., Miller, S., and Koltun, V. A large dataset of object scans. *arXiv:1602.02481* (2016).
10. Cooper, M. *Line Drawing Interpretation*. Springer-Verlag London, 2008.
11. De Paoli, C., and Singh, K. Secondskin: Sketch-based construction of layered 3d models. *ACM Trans. Graph. 34*, 4 (July 2015), 126:1–126:10.
12. Dollár, P., and Zitnick, C. L. Fast edge detection using structured forests. *IEEE transactions on pattern analysis and machine intelligence 37*, 8 (2015), 1558–1570.
13. Dorsey, J., Xu, S., Smedresman, G., Rushmeier, H., and McMillan, L. The mental canvas: A tool for conceptual architectural design and analysis. In *PG '07* (2007).
14. Eitz, M., Richter, R., Boubekeur, T., Hildebrand, K., and Alexa, M. Sketch-based shape retrieval. *ACM Trans. Graph. 31*, 4 (July 2012), 31:1–31:10.
15. Engel, J., Koltun, V., and Cremers, D. Direct sparse odometry. In *arXiv:1607.02565* (July 2016).
16. Favreau, J.-D., Lafarge, F., and Bousseau, A. Line drawing interpretation in a multi-view context. In *CVPR* (2015).
17. Gannon, M., Grossman, T., and Fitzmaurice, G. Tactum: a skin-centric approach to digital design and fabrication. In *CHI '15* (2015), 1779–1788.
18. Gingold, Y., Igarashi, T., and Zorin, D. Structured annotations for 2D-to-3D modeling. *ACM Transactions on Graphics (TOG) 28*, 5 (2009), 148.
19. Google, Inc. Tilt brush. 2015.
20. Holz, D., Holzer, S., Rusu, R. B., and Behnke, S. Real-time plane segmentation using rgb-d cameras. In *Robot Soccer World Cup*, Springer (2011), 306–317.
21. Huo, K., Vinayak, and Ramani, K. Window-shaping: 3d design ideation in mixed reality. In *SUI '16* (2016), 189–189.
22. Igarashi, T., and Hughes, J. F. A suggestive interface for 3d drawing. In *UIST '01* (2001), 173–181.
23. Igarashi, T., Matsuoka, S., and Tanaka, H. Teddy: A sketching interface for 3d freeform design. In *SIGGRAPH '99* (1999), 409–416.
24. Isack, H., and Boykov, Y. Energy-based geometric multi-model fitting. *Int. J. Comput. Vision 97*, 2 (Apr. 2012), 123–147.
25. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *UIST '11* (2011), 559–568.
26. Janoch, A., Karayev, S., Jia, Y., Barron, J. T., Fritz, M., Saenko, K., and Darrell, T. A category-level 3d object dataset: Putting the kinect to work. In *Consumer Depth Cameras for Computer Vision*. Springer, 2013, 141–165.
27. Jones, W., and Sagoo, N. *Architects' Sketchbooks*. Thames and Hudson, 2011.
28. Kallio, K. 3D6B Editor: Projective 3D Sketching with Line-Based Rendering. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2005).
29. Kim, Y., and Bae, S.-H. Sketchingwithhands: 3d sketching handheld products with first-person hand posture. In *UIST '16* (2016), 797–808.
30. Kolmogorov, V., and Zabih, R. What energy functions can be minimized via graph cuts? In *ECCV '02* (2002), 65–81.
31. Lau, M., Saul, G., Mitani, J., and Igarashi, T. Modeling-in-context: User design of complementary objects with a single photo. In *Proc. SBIM* (2010), 17–24.
32. Lipp, M., Wonka, P., and Müller, P. Pushpull++. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 130.
33. Lipson, H., and Shpitalni, M. Optimization-based reconstruction of a 3d object from a single freehand line drawing. In *ACM SIGGRAPH 2007 courses*, ACM (2007), 45.
34. Liu, F., Shen, C., Lin, G., and Reid, I. Learning depth from single monocular images using deep convolutional neural fields. *IEEE transactions on pattern analysis and machine intelligence 38*, 10 (2016), 2024–2039.

35. Monszpart, A., Mellado, N., Brostow, G. J., and Mitra, N. J. Rapter: Rebuilding man-made scenes with regular arrangements of planes. *ACM Trans. Graph. 34*, 4 (July 2015), 103:1–103:12.

36. Nathan Silberman, Derek Hoiem, P. K., and Fergus, R. Indoor segmentation and support inference from rgbd images. In *ECCV* (2012).

37. Nealen, A., Igarashi, T., Sorkine, O., and Alexa, M. FiberMesh: Designing freeform surfaces with 3D curves. *ACM Transactions on Graphics 26*, 3 (2007), article no. 41.

38. Nuernberger, B., Ofek, E., Benko, H., and Wilson, A. D. Snaptoreality: Aligning augmented reality to the real world. In *CHI '16* (2016), 1233–1244.

39. Olsen, L., Samavati, F. F., Sousa, M. C., and Jorge, J. A. Sketch-based modeling: A survey. *Computers & Graphics 33*, 1 (2009), 85–103.

40. Owada, S., Nielsen, F., Nakazawa, K., and Igarashi, T. A sketching interface for modeling the internal structures of 3d shapes. In *ACM SIGGRAPH 2007 courses*, ACM (2007), 38.

41. Öztireli, A. C., Uyumaz, U., Popa, T., Sheffer, A., and Gross, M. 3d modeling with a symmetric sketch. EG (August 2011).

42. Paczkowski, P., Dorsey, J., Rushmeier, H., and Kim, M. H. Paper3d: Bringing casual 3d modeling to a multi-touch interface. In *UIST '14* (2014), 23–32.

43. Paczkowski, P., Kim, M. H., Morvan, Y., Dorsey, J., Rushmeier, H., and O'Sullivan, C. Insitu: Sketching architectural designs in context. *ACM TOG (SIGGRAPH Asia) 30*, 6 (2011), 182:1–10.

44. Sachs, E., Roberts, A., and Stoops, D. 3draww: A tool for designing 3d shapes. *IEEE Comput. Graph. Appl. 11*, 6 (Nov. 1991), 18–26.

45. Sadri, B., and Singh, K. Flow-complex-based shape reconstruction from 3d curves. *ACM Trans. Graph. 33*, 2 (Apr. 2014), 20:1–20:15.

46. Schmidt, R., Khan, A., Singh, K., and Kurtenbach, G. Analytic drawing of 3d scaffolds. In *ACM TOG (SIGGRAPH Asia)*, vol. 28 (2009), 149.

47. Schnabel, R., Wahl, R., and Klein, R. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum 26*, 2 (June 2007), 214–226.

48. Shao, C., Bousseau, A., Sheffer, A., and Singh, K. Crossshade: Shading concept sketches using cross-section curves. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings) 31*, 4 (2012).

49. Shao, T., Li, W., Zhou, K., Xu, W., Guo, B., and Mitra, N. J. Interpreting concept sketches. *ACM TOG (SIGGRAPH) 32*, 4 (2013).

50. Shao, T., Monszpart, A., Zheng, Y., Koo, B., Xu, W., Zhou, K., and Mitra, N. J. Imagining the unseen: Stability-based cuboid arrangements for scene understanding. *ACM Transactions on Graphics 33*, 6 (2014), 209:1–209:11.

51. Shao, T., Xu, W., Zhou, K., Wang, J., Li, D., and Guo, B. An interactive approach to semantic modeling of indoor scenes with an rgbd camera. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 136.

52. Shen, C.-H., Fu, H., Chen, K., and Hu, S.-M. Structure recovery by part assembly. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2012 31*, 6 (2012), 180:1–180:11.

53. Shesh, A., and Chen, B. Smartpaper: An interactive and user friendly sketching system. In *Computer Graphics Forum*, vol. 23, Wiley Online Library (2004), 301–310.

54. Shtof, A., Agathos, A., Gingold, Y., Shamir, A., and Cohen-Or, D. Geosemantic snapping for sketch-based modeling. *Computer Graphics Forum 32*, 2 (2013), 245–253.

55. Snavely, N., Seitz, S. M., and Szeliski, R. Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH Conference Proceedings*, ACM Press (New York, NY, USA, 2006), 835–846.

56. Song, S., and Xiao, J. Sliding shapes for 3d object detection in depth images. In *European Conference on Computer Vision*, Springer (2014), 634–651.

57. Thormählen, T., and Seidel, H.-P. 3d-modeling by ortho-image generation from image sequences. In *ACM Transactions on Graphics (TOG)*, vol. 27, ACM (2008), 86.

58. Trimble, Inc. SketchUp. 2017.

59. Xie, X., Xu, K., Mitra, N. J., Cohen-Or, D., Gong, W., Su, Q., and Chen, B. Sketch-to-design: Context-based part assembly. *Comput. Graph. Forum 32*, 8 (2013), 233–245.

60. Xu, B., Chang, W., Sheffer, A., Bousseau, A., McCrae, J., and Singh, K. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM TOG (SIGGRAPH) 33*, 4 (2014).

61. Xu, K., Chen, K., Fu, H., Sun, W.-L., and Hu, S.-M. Sketch2scene: Sketch-based co-retrieval and co-placement of 3d models. *ACM Transactions on Graphics 32*, 4 (2013), 123:1–123:15.

62. Zeleznik, R. C., Herndon, K. P., and Hughes, J. F. Sketch: An interface for sketching 3d scenes. In *SIGGRAPH '96* (1996), 163–170.

63. Zhang, Y., Xu, W., Tong, Y., and Zhou, K. Online structure analysis for real-time indoor scene reconstruction. *ACM Transactions on Graphics (TOG) 34*, 5 (2015), 159.

64. Zheng, Y., Chen, X., Cheng, M.-M., Zhou, K., Hu, S.-M., and Mitra, N. J. Interactive images: Cuboid proxies for smart image manipulation. *ACM Trans. Graph. 31*, 4 (July 2012), 99:1–99:11.

65. Zheng, Y., Liu, H., Dorsey, J., and Mitra, M. Smart canvas: Context-inferred interpretation of sketches for preparatory design studies. *Computer Graphics Forum (Proc. Eurographics) 35*, 2 (2016).

66. Zollhöfer, M., Nießner, M., Izadi, S., Rehmann, C., Zach, C., Fisher, M., Wu, C., Fitzgibbon, A., Loop, C., Theobalt, C., et al. Real-time non-rigid reconstruction using an rgb-d camera. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 156.