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

Yuwei Li¹  Xi Luo¹  Youyi Zheng²,¹  Pengfei Xu³  Hongbo Fu¹

¹ShanghaiTech Univ.  ²State Key Lab of CAD&CG, Zhejiang Univ.  ³Shenzhen Univ.
⁴City Univ. of Hong Kong

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

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.

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 these 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 enabling 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.

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 lightweight, 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-canvas and a candidate panel where best-ranked candidate s-canvas 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-canvas will be shown 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 is 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

Figure 4. Various cases to create new s-canvas: (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.
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-canvases 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 the plane to a desired position for stroke drawing. See Figure 6 for an example.

**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)).

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 clus-
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 \( \Omega \). 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 \( \Omega \). 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.

**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: attach 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]. 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.

**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-canvases 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 \( \Omega = \{p_1, p_2, \ldots, p_n\} \). 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:

\[
\argmin_{i,j} E_{i,j} := \left\{ \lambda_p E_U(s_p \rightarrow p_i) + \lambda_t E_U(s_t \rightarrow p_j) \right\} + \lambda_s E_B(s_p \rightarrow p_i, s_t \rightarrow p_j),
\]

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:

\[
E_U(s \rightarrow p) = \psi(s \mid p) \otimes \eta(s),
\]

where \( \psi(s \mid p) \) is the likelihood of a stroke \( s \) lying on the plane \( p \), defined as \( e^{-\tau(s \mid p)^2/2\sigma_p^2} \) 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). \( \sigma_p \) 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 snapping results, while on the other hand to encourage the selection of planes which are approximately orthogonal. We define it as follows:

\[
E_B(s_p \rightarrow p_i, s_t \rightarrow p_j) = e^{-\theta_{i,j}/2\sigma^2},
\]

where \( \theta_{i,j} \) is the angle (with the range from 0 to \( \pi/2 \)) between planes \( p_i \) and \( p_j \), and \( \sigma \) 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.

**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 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 workspace, 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 ~ 20 minutes to create each example, including the idling time. All examples were created on a Laptop with an Intel(R) Core(TM) i7-2620M 2.7GHz CPU and 8GB RAM.

<table>
<thead>
<tr>
<th>Stat.</th>
<th>truck</th>
<th>Chn style</th>
<th>piano</th>
<th>kitchen</th>
<th>console</th>
<th>hallway</th>
<th>time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#canvas</td>
<td>21</td>
<td>26</td>
<td>31</td>
<td>48</td>
<td>48</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>time (min)</td>
<td>6</td>
<td>8</td>
<td>15</td>
<td>15</td>
<td>29</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

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

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.

**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.

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 pro-
cess. 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).

**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

---

**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 $1^\circ$ 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.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Method</th>
<th>Mean (SD)</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stair</td>
<td>RANSAC</td>
<td>4.01° (2.10°)</td>
<td>14.28%</td>
<td>66.67%</td>
</tr>
<tr>
<td></td>
<td>Monszpart</td>
<td>3.16° (3.64°)</td>
<td>44.44%</td>
<td>91.67%</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>2.87° (3.77°)</td>
<td>50.00%</td>
<td>92.39%</td>
</tr>
<tr>
<td>Chair</td>
<td>RANSAC</td>
<td>4.73° (5.40°)</td>
<td>33.33%</td>
<td>67.18%</td>
</tr>
<tr>
<td></td>
<td>Monszpart</td>
<td>2.00° (2.39°)</td>
<td>25.00%</td>
<td>90.62%</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>2.61° (3.01°)</td>
<td>62.50%</td>
<td>90.60%</td>
</tr>
</tbody>
</table>

---

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

**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 ~ 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 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
In the existing context is used largely depends on how many re-
lations 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,
e tc. 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 pho-
tograph, 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 an-
other choice but might exclude the user from an experience
of modeling in “reality”. Alternatively, one could also lever-
age 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 mod-
eling furniture-level objects in indoor scenes due to the acquis-
tion capability of consumer-level depth sensors. While the
resulting models are relatively low-resolution, the high ex-
pressiveness of our tool can be easily seen from the examples
and it is useful for concept design.

More detailed modeling features such as profile/trajectory
stroke editing, surface sculpting, extrusion, sharpening, 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 view-
ing direction, leading to effects such as foreshortening. Our
system will omit such long projection as it is undesirable. Ad-
ditionally, 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 cas-
es, the user needs to perform the auxiliary operation of “plane
manipulation” to manually create such a 3D plane. Our sys-
tem currently only exploits planar structure of the underlying
scenes. More complex relationship such as repetition, spac-
ing, or curved surfaces are not considered. Although they
can be extended in general, however, extracting such infor-
mation 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 convention-
al 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 formu-
late 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 large-
ly exploits the RGB-D information in the context and lever-
ages planar structure analysis to help anchor the user drawn
strokes into 3D to create swept surfaces. The core of our for-
mulation 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, orthog-
onal 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 Chi-
na Young 1000 Talents Program, Shenzhen Innovation Pro-
gram NO. JCYJ20170302154106666, and the grants from
the Research Grants Council of the Hong Kong Special Ad-
ministrative Region, China (Project No. CityU11300615, C-
tityU11204014, 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.


