Learning assisted Interactive Modelling with Rough Freehand 3D Sketch Strokes

Sukanya Bhattacharjee
IIT Bombay
India
sukanyabhat@cse.iitb.ac.in

Parag Chaudhuri
IIT Bombay
India
paragc@cse.iitb.ac.in

Abstract

Freehand 3D sketches are a great medium to ideate and create visual content. However, generating 3D models from such rough sketches remains an unsolved, non-trivial task. We present an end-to-end interactive framework for rapid, incremental modelling from sparse, irregular 3D sketches. At the core of our solution, is sketchTransformer*, a two-staged transformer network architecture, that fits parametric surface patches to a set of sketch strokes. We devise a novel pseudo height field representation that enables the sketchTransformer to handle noise and sparseness in the input strokes. Our method interactively evolves the surface model while maintaining smooth joins between nearby patches. We implement two frontends for our framework, one on the desktop and another as a mobile AR application, to illustrate how our method complements a standard 3D modelling pipelines. Our framework robustly handles a large variety of input 3D strokes that competing methods cannot parse adequately.*

1. Introduction

Sketches are an intuitive form of visual ideation and an invaluable design tool for rapid pictorial depiction of 3D objects. Sketches traditionally have been restricted to 2D, so translating concept sketches to 3D requires additional work. With the advancements in Augmented/Virtual Reality (AR/VR) technologies, sketching directly in 3D space is becoming more accessible [4]. However, sketching in a free 3D space is inherently troublesome because of the absence of any tactile feedback. This makes it difficult for the artist to consistently place sketch strokes on intended surfaces.

We present an learning assisted interactive framework to create 3D surface models from rough freehand 3D sketches drawn incrementally. At the core of our framework is a transformer based architecture, *sketchTransformer*, that fits parametric (BSpline) surface patches to a sparse collections of 3D sketch strokes. The performance of this neural architecture is crucially enhanced by our novel pseudo-height field representation allowing it to robustly handle the sparseness and noise in the input. As the user draws new strokes, our framework tries to either fit a new patch to the sketched strokes, or edit and re-fit existing surface patches based upon the inferred sketch semantics. The surface patches are joined smoothly or kept disjoint as per the intent.

Our method works with a variety of freehand 3D sketches, while adapting to their inherently unstructured nature. Therefore, it performs significantly better than the state-of-the-art methods that can perform similar tasks (as shown in Section 5.1). However, this robustness comes at the cost of fine control during modelling. While our framework is apt for rapidly creating surface models from 3D sketches, it is not intended to be used for finely nuanced modelling. Note that our framework is aimed at creating surface models incrementally from freehand 3D sketches directly. We show two prototype implementations of our framework, one on the desktop as a Blender [7] plugin and the other in AR on a mobile device. In order to assess the quality of the models created using our framework, we present an extensive user study (in Section 5.4) which shows that our interactive framework is a valuable tool for creative ideation with 3D sketches.

Our main contributions are:

- An end-to-end interactive neural framework for rapid, incremental modelling from sparse, irregular 3D sketches.
- Frontends on desktop and mobile AR complementing the standard 3D modelling pipelines.

2. Related Work

A lot of prior work deals with 3D modeling from sketches. Early works like [27] inspired from [18] enable modifying a 3D models interactively with control curves while creating it with 2D sketches only. Works like [9, 21] create 3D models from 2D sketches input from multiple views. Although, these methods process the depth information extracted from the multi-view input but do not enable 3D sketching. Other works like [2, 14, 35] provide methods to process sparse freehand 2D sketches to create 3D models. These meth-

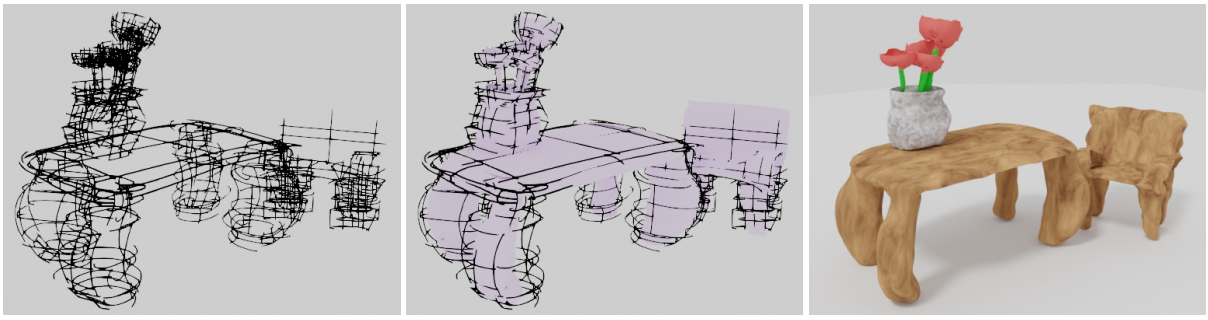

Figure 1. 3D sketches of multiple objects made using our framework are assembled into a scene. The images from left to right show the sketch strokes, the reconstructed surface model made of parametric patches and the rendered meshes created from the sketches.

ods allow incremental modelling from sketches and editing the output, still do not handle 3D sketches. Complementary to these works, [13] transforms a vector 2D sketch into a coherent 3D sketch to output models which can be interactively modified.

Works like [1, 8, 37] devise reference model or canvas guided 3D sketching method to create 3D models. This approach restricts the modelling to the reference models or canvas to be used. Other category of works like [29] directly utilize the 3D sketch strokes drawn in VR to create manifold surface strips stitched together to form the 3D model. These methods require a large number of input strokes. More recent works like [30, 40] process the input VR sketch strokes in a way that they become suitable for modelling with existing algorithms.

Most recently, [41] provide a framework which creates 3D model from 3D sketches as a collection of surfaces smoothly joined together, given an initial surface model for the input. This approach assures high accuracy but heavily depends on availability of the initial model. Similarly, [36] provide method to create 3D models from rough 3D mid-air sketches with available reference point cloud to assist in the sketching.

Methods like [17, 38] fit surface models to a sparse unoriented 3D point sets by computing their normals first and then extracting the surface via constrained optimizations. These approaches cannot work at real-time. Other category of works like [16, 25, 31, 39] learn to reconstruct 3D models from 3D point clouds via a different neural networks based upon auto-encoders, neural kernels, etc. Since these methods are designed for point clouds, they require a uniformly sampled point set as opposed to unstructured sketches.

Works like [23, 24, 26, 32, 42] devise various neural architectures to interpret 3D sketches. These methods target to model only a particular class of objects. Works like [5] design neural networks which learn to fit a surface to stream of incoming 3D sketch strokes. This method is able to interactively modify surface models but heavily depend on availability of the boundary information of the surface patches. Other works like [22] translate clean freehand drawings into procedural commands which are used to generate the corresponding CAD model. This method does not deal with surface modelling of freeform objects.

While these methods work well in their specific contexts, they struggle to create 3D models with rough freehand 3D sketches. Also, most of these methods do not support interactivity in the modelling process which we intend to provide.

3. Method

Our framework for interactively creating surface models from 3D sketches comprises of three components (see Figure 2). We describe each of these components in the subsequent subsections.

3.1. Pre-processing

The pre-processing component of our framework takes the raw sketch strokes, clusters and encodes them into a more regularized representation.

3.1.1 Contextual Segmentation of Sketch Strokes

As the user starts to sketch, our framework contextually segments the sketch strokes into clusters, such that we can fit a parametric surface patch to each cluster. The algorithm for contextual stroke segmentation is given in Algorithm 1.

Every stroke sketched by the user can be one of two kinds:

Shaping strokes: These are the strokes which define the shape of the surface to be fit. The blue stroke in the first image in Figure 3 is an example of a shaping stroke.

Joining strokes: These are the strokes which span more than one surface patch and determine how the surfaces join with each other. Note that more than one joining stroke may define the join between a pair of surfaces. The red stroke in the first image in Figure 3 is an example of a joining stroke.

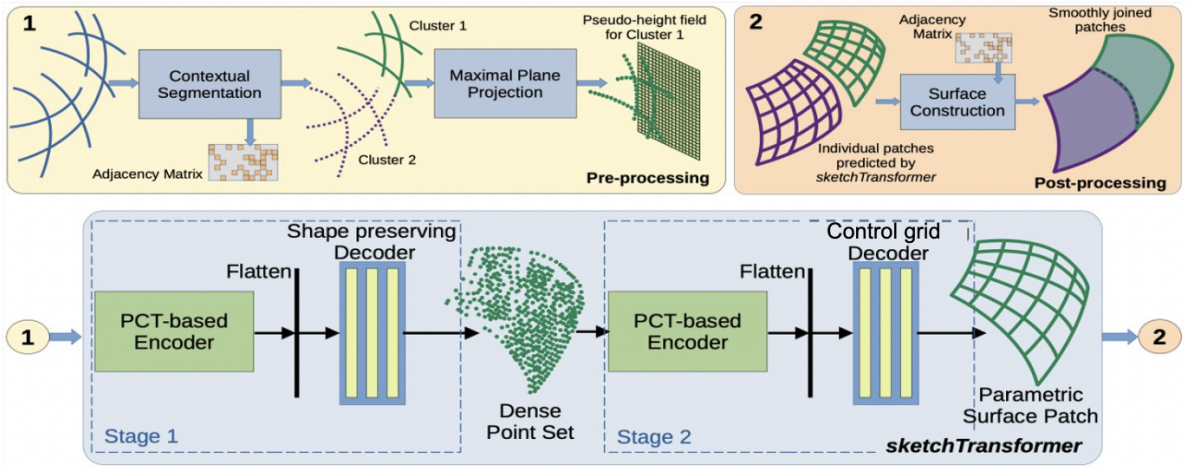

Figure 2. Overview of our framework: (1) Pre-processing, (2) Post-processing, and the core **sketchTransformer** architecture.

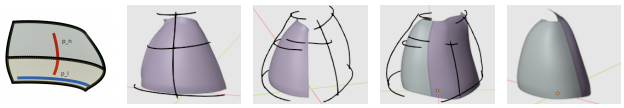

Figure 3. First image shows the types of strokes - shaping stroke in blue, joining stroke in red. Subsequent images show how a new surface patch gets added and an existing patch is modified when a joining stroke is input. The fourth image shows the new patch created from the sketch strokes shown in the third image. The last image shows the change in the existing patch from the sketch strokes shown in the third image.

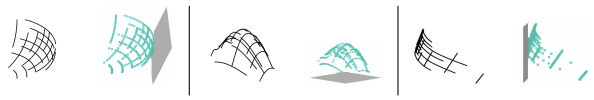

Figure 4. Example pairs of sketch strokes and their pseudo-height field. The plane in gray shows the corresponding maximal plane.

In Algorithm 1, $\mathcal{C}(.)$ is the 3D convex hull. $dist(.)$ computes the distance between the centroids of the two strokes. We empirically determine the value for the distance threshold, τ_d. Initially, P and C are empty sets, and the adjacency matrix A' is a zero matrix. At the end of Algorithm 1, the new clusters in C have no surface patches corresponding to them. These patches are created in subsequent parts of the framework. The existing surface patches may also need to be updated if there is some addition to their corresponding clusters in the algorithm. Since we sample points from the strokes and then cluster them, the algorithm is independent of the stroke style. Figure 3 shows how strokes modify a surface patch and create a new one that smoothly joins with the existing patch. The output of Algorithm 1 is a set of updated stroke clusters C and the updated adjacency matrix A' for all the pairs of patches in the model.

3.1.2 Maximal Axial Plane Projection

We sample points from every stroke from C, and then encode these point samples into a more regularized representation which suitable for processing by the network that follows. We require a representation which is not only more uniform that the raw input but also conveys some information about the extent of the target shape.

To create this representation, we project the strokes on each axial plane ($\mathbb{XY}$, $\mathbb{YZ}$, $\mathbb{ZX}$) in turn, and retain the plane (and corresponding projections) which has the maximum coverage of the points in the projection. This ensures that majority of points from the sketch are retained in the projection. We call this as *maximal axial plane projection*. We sample a dense regular grid of points on the retained axial plane. We empirically choose a 64×64 grid size on this planes for this purpose. For every projected point, we choose the grid point that is closest to the projected point and associate the distance of the original stroke sample from its projection to that grid point sample. Keeping all other points with the value 0 helps to give an idea of the actual extent of the intended shape to the network. This creates a *pseudo-height field* of the stroke sample point set bound by the chosen plane. Examples of this can be seen in Figure 4. The *pseudo-height field* makes our network robust to the number of input strokes and induces a structure to the input.

The input to this entire pre-processing component is a set of strokes $S = \{s_k\}_{k=1}^{N}$. Every stroke s_k is a set of points $\{q_l\}_{l=1}^{Q_k}$, where $q_l \in \mathbb{R}^3$. From this, a set of stroke clusters $C = \{c_j\}_{j=1}^{M}$ is output by Algorithm 1. Here, c_j is a set of strokes $\{s_{k_j}\}_{k_j=1}^{M_j}$. Each c_j is converted to the *pseudo-height field*, h_j, as given in section 3.1.2.

Algorithm 1 Contextual Segmentation of Strokes

Let the set of current surface patches be P
Let the set of corresponding stroke clusters be C
Let A' be the current adjacency matrix.
Fetch currently sketched stroke s_i
$AdjPatch \leftarrow -1$
for Patches $p_j \in P$ **do**
  $CurrentPatch \leftarrow j$
  if s_i is completely in $\mathcal{C}(p_j)$ **then**
    s_i is a shaping stroke for existing patch
    Add s_i to stroke cluster c_j, corresponding to p_j
    if $Adjpatch \neq -1$ **then**
      Update $A'[AdjPatch, CurrPatch] = 1$
      $AdjPatch \leftarrow -1$
    end if
  else if s_i is partially in $\mathcal{C}(p_j)$ **then**
    s_i is a joining stroke
    Split s_i into $s_i^a \in \mathcal{C}(p_j)$ and $s_i^b \notin \mathcal{C}(p_j)$
    Add s_i^a to stroke cluster c_j, corresponding to p_j
    Set $s_i \leftarrow s_i^b$
    $AdjPatch \leftarrow j$
  else
    if $(\exists s_i' : s_i' \notin c_k \forall k)$ AND (dist(s_i, s_i') ¡ τ_d) **then**
      Add s_i and s_i' into new cluster c_j'
      Add c_j' to C
    end if
  end if
end for
return C, A'

3.2. The sketchTransformer Network

We design our core neural network to fit a BSpline surface patch to each pseudo-height field, h_j. In order to do this, it is important to capture the influence of each point from h_j on the target shape. This is done by adding the attention mechanism [34] via a transformer [10] based network. We use a Point Cloud Transformer (PCT) [15] inspired encoder. We found transformers are better equipped to handle the unstructured and irregular set of points as compared to the CNNs. We cascade two encoder-decoder stages to obtain the required parametric surface patch for given *pseudo-height field* (see bottom image in Figure 2). We design a two-staged network as opposed to a single one as we found the latter to fail to get accurate results. It becomes too difficult for the network to learn to infer the intended shape from an input which is heavily sparse and irregular. The point set obtained after the Stage 1 network is significantly denser and regular, hence the control grids obtained from it are of higher quality.

3.2.1 Stage 1 - Dense Point Set Construction

Each pseudo-height field, h_j, (4096×3) is input to this network. The encoder outputs the positional, maximal and average feature sets for given set of points (1024×3). We concatenate these feature sets into a flattened vector and then pass it on to our decoder. The decoder outputs a dense set of points on the intended surface patch represented by the input strokes set. This transformed output point set r_k has 1024 points. Note that we get a denser set of points in regions where there is enough information obtainable from the input strokes, while other regions are relatively sparser. Thus, the spatial distribution of the points on the surface patch remains irregular and noisy. The point set, r_j, is input to the next encoder-decoder stage where a BSpline surface patch is fit to it.

3.2.2 Stage 2 - Parametric Surface Patch Fitting

The architecture of Stage 2 of sketchTransformer is identical to Stage 1, with two key differences. We only use the average feature set from the output of the encoder to combat the effects of noise by summarizing the neighborhood of each point while computing the attention on it. This feature set is passed to our decoder. In contrast to Stage 1 decoder, this decoder downsamples the input, to output a control grid of size 16×16. We compute a BSpline surface of degree 6 (in both u and v directions) from this control grid with clamped uniform knot vectors. We call this *surface construction*.

This stage outputs a control grid g_j corresponding to a BSpline surface patch p_j for each input h_j. The final output is a set of control grids, $G = \{g_j\}_{j=1}^{M}$.

3.3. Post-processing

A complete surface model is assembled with this post-processing component. The set of control grids G and the adjacency matrix A'(computed in Section 3.1.1) is input to this component. Since at the end of Algorithm 1 there are no surface patches for the new stroke clusters, we update their adjacency to get the final adjacency matrix A. We join the adjacent surface patches such that local continuity is preserved among them. For this, we identify their corresponding adjacent boundaries by finding the boundaries intersected by the joining stroke. The boundary control points of both surfaces are adjusted and stitched together. Figure 3 shows examples of how our framework modifies a previously fit surface while connecting it with the newly created surface.

4. Training sketchTransformer

We train the core sketchTransformer network with a dataset containing pairs of 3D sketch strokes and surface patches.

We extract BSpline surface patches from the ABC dataset [19]. We sample random curves of varied lengths by choosing consecutive points on the surface patches in either u- or v-directions. We extract points from these curves. We call this dataset as the *ABCSurface* dataset.

For both stages of the network, we use the Chamfer loss [3] (Eq. 1) between point set sampled from the predicted surface patch P and point set sampled from the corresponding ground truth surface patch S.

$$\mathcal{L}_{\text{surf}} = \max\left(\frac{1}{|S|}\sum_{x\in S}\min_{\hat{x}\in P}||x-\hat{x}||_2, \frac{1}{|P|}\sum_{\hat{x}\in P}\min_{x\in S}||\hat{x}-x||_2\right) \tag{1}$$

For the Stage 2 network, we use two additional loss terms. We use the L2 loss (Eq. 2) between the ground truth control grid S_{cp} and the control grid computed by our method P_{cp}.

$$\mathcal{L}_{\text{CP}} = \frac{1}{|S_{cp}|}\sum_{x\in S_{cp}}||x_{cp}-\hat{x}_{cp}||_2 \tag{2}$$

Taking inspiration from [31], we compute the L2 loss (Eq. 3) between the Laplacian of the output grid P_{cp} and the best oriented output grid P_{cp^*}.

$$\mathcal{L}_{\text{lap}} = \frac{1}{|P_{cp}|}\sum_{\hat{x}\in P_{cp}}||lap(\hat{x}_{cp^*})-lap(\hat{x}_{cp})||_2 \tag{3}$$

5. Results

5.1. Comparison with Baselines

We use the Chamfer (Eq. 1) and directed Hausdorff (Eq. 4) distances as the error metrics for all the comparisons. P is the predicted surface patch and S is the corresponding ground truth surface patch.

$$d_h(P,S) = \max_{x\in S}\min_{\hat{x}\in P}||x-\hat{x}||_2 \tag{4}$$

5.1.1 Single Surface Patch Fitting Evaluation

We perform all these evaluations on the ABCSurface dataset and train the neural baselines with same set-up as ours. We compare the performance of our method sketchTransformer (SK) with both parametric and implicit surface fitting baselines. Both these types of surface patches are capable of representing surface geometry. For the parametric fitting baselines, we compare with the least squared [6] (LSQ) based approximation of control grid from a given set of points, neural based method ParseNet [31] (PN) which predicts control grid from a set of points, and neural based method StPNet [5] (SN) which predicts control grid from a set of 3D strokes. For the implicit fitting baselines, we compare with a non-neural method VIPSS [17] which fits surface to a given set of sparse points, and a neural method NKSR [16] which fits surface to sparse and noisy set of points. Table 1 and Figure 5 shows that our sketchTransformer network outperforms all the baselines for the single patch fitting task.

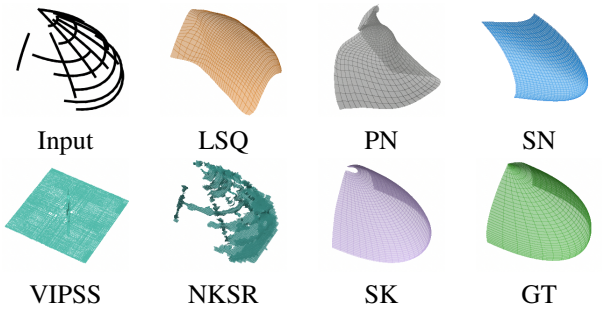

Figure 5. Comparing with baselines for creating single surface patches from input strokes, on ABCSurface dataset. **LSQ:** [28], **PN:** [31], **SN:** [5], **VIPSS:** [17], **NKSR:** [16], **SK:** Our method.

	LSQ	PN	SN	VIPSS	NKSR	SK
Cfr	0.441	0.066	0.379	0.136	0.031	**0.024**
Hdf	0.377	0.572	1.101	0.299	0.299	**0.298**

Table 1. Comparing with baselines for creating single surfaces from input strokes on ABCSurface dataset. **Cfr:** Chamfer Error, **Hdf:** Hausdorff Error, **LSQ:** [28], **PN:** [31], **SN:** [5], **VIPSS:** [17], **NKSR:** [16], **SK:** Our method.

5.1.2 Multiple Surface Patch Fitting Evaluation

We evaluate our framework for multiple surface fitting on the standard patch based models from the classic Utah tea set dataset [33] comprising of teapot, teacup and teaspoon models. We extract the bicubic Bezier patches from the extended set of Utah teapot models and sample strokes on these patches. Since LSQ [6] and Parsenet [31] are not equipped to create complete 3D models with only surfaces, we compare with the VIPSS [17] and NKSR [16] baselines which are capable of creating surface models. Additionally, we compare with a non-neural method [41] (SF) which fits piece-wise smooth surfaces given a set of 3D strokes and an initial surface model. Note that all the three baselines output an implicit surface model whereas our method outputs a surface model comprising of a collection of BSpline surface patches. So, we sample points from each of these surface models to compute their error with respect to the ground truth models. Table 2 shows the average error between points sampled from predicted patches and ground truth patches for the Utah teapot models. First two columns of Figure 6 shows qualitative results for the same. Since our method supports modelling from sketch strokes incrementally added, we also compare the performance of our

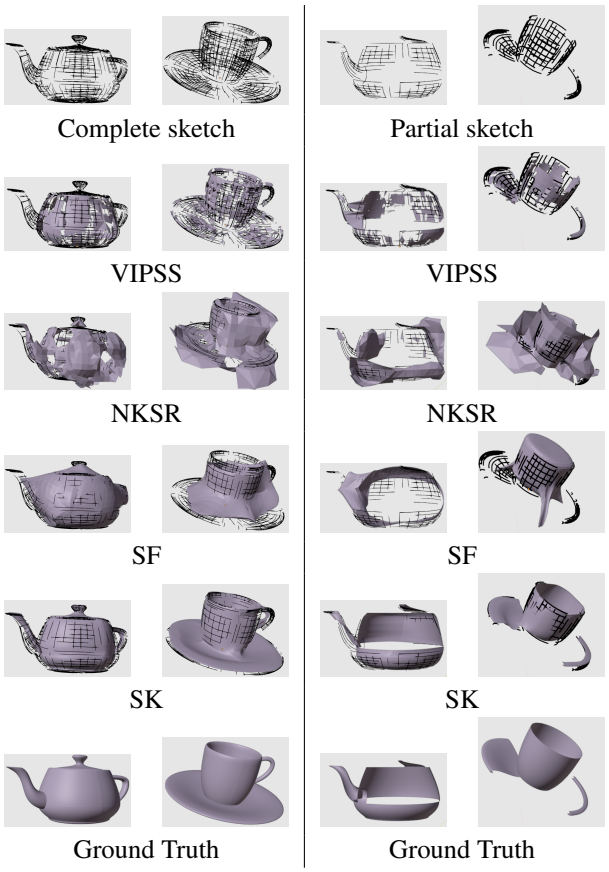

Figure 6. Qualitative comparison with baselines for surface modelling from complete and partial sketch strokes sampled from two models of the Utah tea set dataset [33]. **VIPSS:** [17], **NKSR:** [16], **SF:** [41], **SK:** Our method

		Teapot	Teacup	Teaspoon
Chamfer	VIPSS	0.3362	0.9473	0.6791
	NKSR	0.1163	0.1811	0.0617
	SF	0.2145	0.933	0.051
	SK	**0.0399**	**0.0982**	**0.0329**
Hausdorff	VIPSS	0.0482	0.3058	0.5286
	NKSR	0.0029	0.0049	0.0019
	SF	0.7456	0.3536	0.0442
	SK	**0.0002**	**0.0002**	**0.00007**

Table 2. Comparison with baselines for surface modelling on the Utah tea set dataset [33] (Teapot, Teacup and Teaspoon models). **VIPSS:** [17], **NKSR:** [16], **SF:** [41], **SK:** Our method.

method with the baselines for partial sketches of the Utah tea set models. Last two columns of Figure 6 shows these results.

5.2. Ablation results

To evaluate the effectiveness of our maximal plane projection, we reconstruct single surface patches using sketch-Transformer with different preprocessing of input. We take a projection on any random plane (SK-Rnd) and on the best aligning plane (SK-Best), and use them as input to sketch-Transformer. We also supply the raw samples from the sketch strokes as input (SK-Raw). The average error across all patches predicted with these three input are shown in Table 3 It can be seen that our framework with the maximal axial plane projection (i.e., SK) performs better than the other choices, for both metrics.

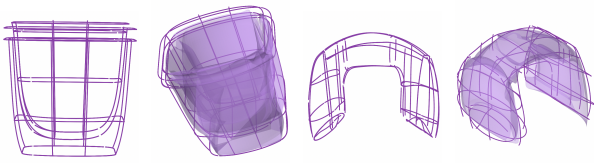

Figure 7. Surface models created with our method from sketches selected from dataset [11]. We show input sketch strokes and the predicted surface models overlaid with the sketch strokes.

	SK-Raw	SK-Rnd	SK-Best	SK
Chamfer	0.0621	0.5538	0.0374	**0.0119**
Hausdorff	0.3856	1.1479	0.3762	**0.2278**

Table 3. Effect of the maximal axial plane projection on sketch-Transformer performance.

5.3. Modelling From Real 3D Sketches

5.3.1 Results on Existing Dataset

We show results of our method on 3D sketches from [11] in Figure 7. These sketches are drawn by artists using various tools designed for 3D sketching. The results show that our method is able to generalize to such sketches and create plausible surface models.

5.3.2 Results of The Blender Frontend

We implement a Blender plugin frontend to our framework as a desktop based modelling application. We use the Grease Pencil [20] tool in Blender to create 3D sketches in real time and our framework to generate the surface models from these sketches. It should be noted that even though a Grease Pencil stroke is planar (on some arbitrary plane), but a collection of these strokes form a non-planar 3D collection of strokes. We show examples for objects created from such sketches in Figure 8. The first column shows the complete 3D sketch, the second shows the parametric surface model and the last shows the recovered meshes.

Figure 1 shows an entire scene that is sketched and created using our Blender frontend. The meshes recovered from the surface models is rendered with applied materials. Figure 9 shows a progression of sketches from the same frontend as the user interactively creates a car model. We

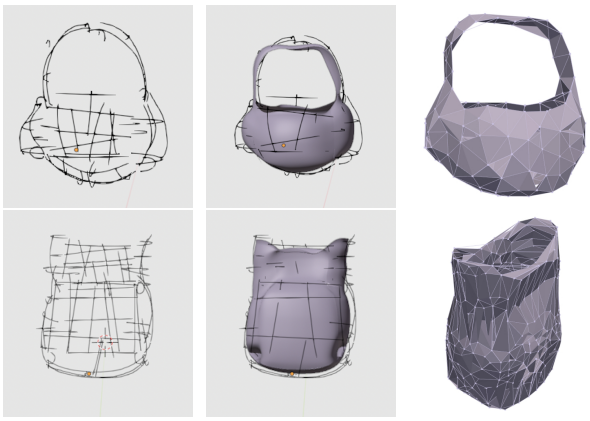

Figure 8. Different 3D models created from real-time 3D sketches with our Blender frontend.

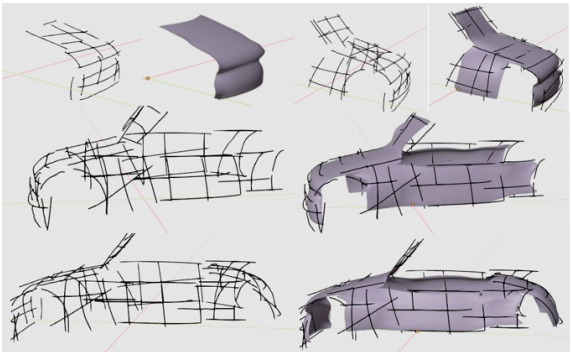

Figure 9. Progressive modelling of an object from 3D sketches with our frontend on desktop in real-time.

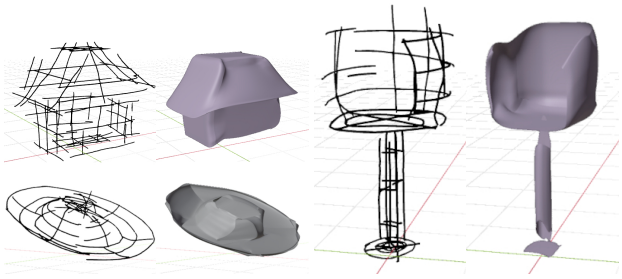

Figure 10. Models created by different users using our frontend implementation on the desktop.

also ask multiple users to try our this frontend and some of the models they created is shown in Figure 10. Top two rows of Figure 13 shows the comparison of our methods and the baselines with these user drawn sketches.

5.3.3 Results of The Mobile AR Frontend

We also implement a Mobile AR frontend of our framework to interactively create models from mid-air sketches in AR.

Figure 11. The first column shows 3D sketches created using the mobile AR frontend. The middle column shows parametric surface models for them created with our framework. Final column shows rendered meshes of the same objects.

Figure 12. Models created by different users using our frontend implementation on mobile AR.

We utilize ARCore [12] for tracking the real world and creating the augmentations. This frontend allows the users to draw in a free, unconstrained 3D space. The user draws the mid-air sketch strokes directly with the phone. They do not have to choose any sketching plane/surface to project onto. The strokes can also be snapped to existing patches to aid in drawing joining strokes. Figure 11 shows various models created interactively with our application from freehand mid-air sketch strokes in AR. The corresponding meshes recovered from the surface models is rendered with applied materials. We also ask multiple users to try this frontend and some of the models they created is shown in Figure 12. Bottom two rows of Figure 13 shows the comparison of our methods and the baselines with these user drawn sketches.

5.4. User Study

We evaluate the 3D models created with our both interactive frontends by showing pairs of 3D sketch and the surface models to 30 participants. Note that we only show the surface models as opposed to the rendered textured meshes to the participants to assess how the raw output is perceived by the users. For each sketch and model pair, the participants

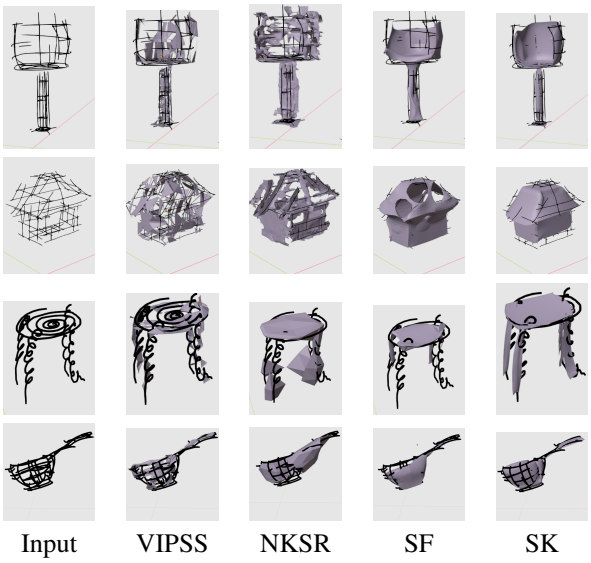

Figure 13. Comparison of modelling with our method and baselines from 3D sketches created by users with both our frontends. **VIPSS:** [17], **NKSR:** [16], **SF:** [41], **SK:** Our method.

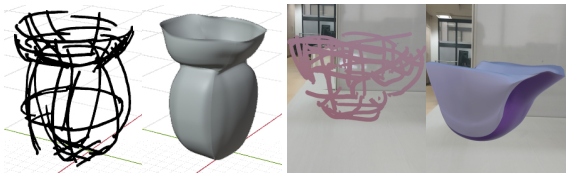

Figure 14. Examples of sketch-model pairs shown during the user study.

are provided with the object's name and a short phrase describing the object. We present 12 such pairs in random order to every participant. Participants are allowed to inspect the sketch and the corresponding parametric surface model in full 3D with Blender or in our mobile AR application. Figure 14 shows two such pairs used in the study. The participants assigned scores for the following questions, in the range 1(worst)-10(best), for each pair shown. The participants were agnostic to the method used to create the models.

- **Q1:** How well constructed/structured is the 3D model?
- **Q2:** How well does the 3D sketch represent the described object?
- **Q3:** How well does the 3D model represent/fit to the 3D sketch?

Figure 15 shows the spread of the scores per question, across all models and participants. It is clear from the study that both the frontends are equally effective as tools for interactive ideation via 3D sketches, and the quality of the models is perceived positively by the participants. Also, many sketches are perceived to be rough or imprecise by the participants, still their models are perceived well.

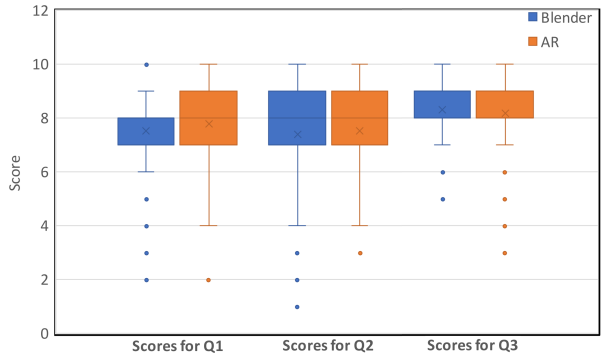

Figure 15. Scores summary from the user study for questions (**Q1**,**Q2**,**Q3**) given section 5.4.

6. Conclusion

We present an interactive framework for incremental creation of models with rough freehand 3D sketches interactively. The core *sketchTransformer* network can robustly fit a parametric surface patch to a sparse and irregular set of 3D sketch strokes. We show a thorough comparative evaluation of our method with various baselines to evaluate its efficacy. We present two frontend implementations for our framework on the desktop and in AR to demonstrate its use for a freeform modelling with 3D sketches. Our method generalizes to real sketch strokes even though it is entirely trained on synthetically sampled sketch strokes. Our underlying patch wise fitting of the given 3D sketches plays a vital role in this. Our core *sketchTransformer* learns a strokes to surface patch mapping which enables it to generalize to distributions different from the training set. Although, we observe from various experiments and the user trials that the network attempts to orient a surface patch with the average orientation of the input 3D sketch strokes.

We deliberately choose not to beautify the strokes sketched by the users. The parametric surface models are also not processed apart from enforcing smooth joins between adjacent surface patches. Hence, the parametric surface models are not watertight and may have holes, depending on the sketches that were used to create them. The user study shows that our framework is a viable choice for rapid ideation and creation of object models via 3D sketches. A limitation of our framework is its unsuitability for detailed or nuanced modelling. In a virtual space, freehand 3D sketches are difficult to control by the user due to lack of any tactile feedback and thus it is difficult for them to give very precise input. Hence, it is not possible to discern finer details from such input directly. As future work, we want to extend our framework for authoring deformable models and animations as well using 3D sketches.

References

[1] Rahul Arora, Rubaiat Habib Kazi, Tovi Grossman, George Fitzmaurice, and Karan Singh. Symbiosissketch: Combining 2d & 3d sketching for designing detailed 3d objects in situ. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 185. ACM, 2018. 2

[2] Hmrishav Bandyopadhyay, Subhadeep Koley, Ayan Das, Ayan Kumar Bhunia, Aneeshan Sain, Pinaki Nath Chowdhury, Tao Xiang, and Yi-Zhe Song. Doodle your 3d: From abstract freehand sketches to precise 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9795–9805, 2024. 1

[3] Harry G Barrow, Jay M Tenenbaum, Robert C Bolles, and Helen C Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. Technical report, SRI International Menlo Park CA AI Center, 1977. 5

[4] Sukanya Bhattacharjee and Parag Chaudhuri. A survey on sketch based content creation: from the desktop to virtual and augmented reality. In *Computer Graphics Forum*, pages 757–780. Wiley Online Library, 2020. 1

[5] Sukanya Bhattacharjee and Parag Chaudhuri. Deep interactive surface creation from 3d sketch strokes. In *Proceedings of IJCAI*, pages 4908–4914. International Joint Conferences on Artificial Intelligence Organization, 2022. AI and Arts Track. 2, 5

[6] Onur Rauf Bingol and Adarsh Krishnamurthy. NURBS-Python: An open-source object-oriented NURBS modeling framework in Python. *SoftwareX*, 9:85–94, 2019. 5

[7] Blender Foundation. Blender 3.6.4 lts, 2023. https://www.blender.org/ Last accessed on 2-10-2023. 1

[8] Chris De Paoli and Karan Singh. Secondskin: sketch-based construction of layered 3d models. *ACM Transactions on Graphics (TOG)*, 34(4):126, 2015. 2

[9] Zhi Deng, Yang Liu, Hao Pan, Wassim Jabi, Juyong Zhang, and Bailin Deng. Sketch2pq: freeform planar quadrilateral mesh design via a single sketch. *IEEE Transactions on Visualization and Computer Graphics*, 2022. 1

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 4

[11] Emilie Yu. 3d sketches curated dataset, 2023. https://gitlab.inria.fr/D3/3d-sketches-curated-dataset Last accessed on 24-02-2024. 6

[12] Google. ARCore, 2020. https://developers.google.com/ar Last accessed on 08-02-2020. 7

[13] Yulia Gryaditskaya, Felix Hähnlein, Chenxi Liu, Alla Sheffer, and Adrien Bousseau. Lifting freehand concept sketches into 3d. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020. 2

[14] Benoit Guillard, Edoardo Remelli, Pierre Yvernay, and Pascal Fua. Sketch2mesh: Reconstructing and editing 3d shapes from sketches. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13023–13032, 2021. 1

[15] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. Pct: Point cloud transformer. *Computational Visual Media*, 7:187–199, 2021. 4

[16] Jiahui Huang, Zan Gojcic, Matan Atzmon, Or Litany, Sanja Fidler, and Francis Williams. Neural kernel surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4369–4379, 2023. 2, 5, 6, 8

[17] Zhiyang Huang, Nathan Carr, and Tao Ju. Variational implicit point set surfaces. *ACM Transactions on Graphics (TOG)*, 38(4):1–13, 2019. 2, 5, 6, 8

[18] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH Conference on Computer graphics and interactive techniques*, pages 409–416. ACM, 1999. 1

[19] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9601–9611, 2019. 5

[20] Joshua Leung and Daniel M Lara. Grease pencil: Integrating animated freehand drawings into 3d production environments. In *SIGGRAPH Asia 2015 Technical Briefs*, pages 1–4. 2015. 6

[21] Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. Robust flow-guided neural prediction for sketch-based freeform surface modeling. *ACM Transactions on Graphics (TOG)*, 37(6):1–12, 2018. 1

[22] Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J Mitra. Free2cad: parsing freehand drawings into cad commands. *ACM Transactions on Graphics (TOG)*, 41(4):1–16, 2022. 2

[23] Zhihao Liu, Fanxing Zhang, and Zhanglin Cheng. Buildingsketch: Freehand mid-air sketching for building modeling. In *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 329–338. IEEE, 2021. 2

[24] Zhongjin Luo, Jie Zhou, Heming Zhu, Dong Du, Xiaoguang Han, and Hongbo Fu. Simpmodeling: Sketching implicit field to guide mesh modeling for 3d animalmorphic head design. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pages 854–863, 2021. 2

[25] Eloi Mehr, Ariane Jourdan, Nicolas Thome, Matthieu Cord, and Vincent Guitteny. Disconet: Shapes learning on disconnected manifolds for 3d editing. In *Proc. of CVPR*, pages 3474–3483, 2019. 2

[26] Suellen Motta, Anselmo Montenegro, Marcelo Gattass, and Deane Roehl. A 3d sketch-based formulation to model salt bodies from seismic data. *Computers & Geosciences*, 142: 104457, 2020. 2

[27] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: designing freeform surfaces with 3d curves. In *ACM SIGGRAPH 2007 papers*, pages 41–50. ACM, 2007. 1

[28] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 1996. 5

[29] Enrique Rosales, Jafet Rodriguez, and Alla Sheffer. Surfacebrush: from virtual reality drawings to manifold surfaces. *ACM Trans. Graph.*, 38(4):96:1–96:15, 2019. 2

[30] Enrique Rosales, Chrystiano Araújo, Jafet Rodriguez, Nicholas Vining, Dongwook Yoon, and Alla Sheffer. Adaptibrush: adaptive general and predictable vr ribbon brush. *ACM Trans. Graph.*, 40(6):247–1, 2021. 2

[31] Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, and Radomír Měch. Parsenet: A parametric surface fitting network for 3d point clouds. In *European Conference on Computer Vision*, pages 261–276. Springer, 2020. 2, 5

[32] Dmitriy Smirnov, Mikhail Bessmeltsev, and Justin Solomon. Deep sketch-based modeling of man-made shapes. *arXiv preprint arXiv:1906.12337*, 2019. 2

[33] Ann Torrence. Martin Newell's original teapot. In *ACM SIGGRAPH 2006 Teapot*, pages 29–es. 2006. 5, 6

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 4

[35] Jiayun Wang, Jierui Lin, Qian Yu, Runtao Liu, Yubei Chen, and Stella X Yu. 3d shape reconstruction from free-hand sketches. In *European Conference on Computer Vision*, pages 184–202. Springer, 2022. 1

[36] Kai Wu and Zhanglin Cheng. Refar: 3d sketch-based modeling with in-situ references. In *2022 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 507–511. IEEE, 2022. 2

[37] Pengfei Xu, Hongbo Fu, Youyi Zheng, Karan Singh, Hui Huang, and Chiew-Lan Tai. Model-guided 3d sketching. *IEEE Transactions on Visualization and Computer Graphics*, pages 2927–2939, 2018. 2

[38] Rui Xu, Zhiyang Dou, Ningna Wang, Shiqing Xin, Shuangmin Chen, Mingyan Jiang, Xiaohu Guo, Wenping Wang, and Changhe Tu. Globally consistent normal orientation for point clouds by regularizing the winding-number field. *ACM Transactions on Graphics (TOG)*, 42(4):1–15, 2023. 2

[39] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 206–215, 2018. 2

[40] Emilie Yu, Rahul Arora, Tibor Stanko, J Andreas Bærentzen, Karan Singh, and Adrien Bousseau. Cassie: Curve and surface sketching in immersive environments. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2021. 2

[41] Emilie Yu, Rahul Arora, J Andreas Baerentzen, Karan Singh, and Adrien Bousseau. Piecewise-smooth surface fitting onto unstructured 3d sketches. *ACM Transactions on Graphics (TOG)*, 41(4):1–16, 2022. 2, 5, 6, 8

[42] Fanxing Zhang, Zhihao Liu, Zhanglin Cheng, Oliver Deussen, Baoquan Chen, and Yunhai Wang. Mid-air finger sketching for tree modeling. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pages 826–834. IEEE, 2021. 2