Sketch2PQ: Freeform Planar Quadrilateral Mesh Design via a Single Sketch

Zhi Deng, Yang Liu, Hao Pan, Wassim Jabi, Juyong Zhang, Bailin Deng[†]

Abstract—The freeform architectural modeling process often involves two important stages: concept design and digital modeling. In the first stage, architects usually sketch the overall 3D shape and the panel layout on a physical or digital paper briefly. In the second stage, a digital 3D model is created using the sketch as a reference. The digital model needs to incorporate geometric requirements for its components, such as the planarity of panels due to consideration of construction costs, which can make the modeling process more challenging. In this work, we present a novel sketch-based system to bridge the concept design and digital modeling of freeform roof-like shapes represented as planar quadrilateral (PQ) meshes. Our system allows the user to sketch the surface boundary and contour lines under axonometric projection and supports the sketching of occluded regions. In addition, the user can sketch feature lines to provide directional guidance to the PQ mesh layout. Given the 2D sketch input, we propose a deep neural network to infer in real-time the underlying surface shape along with a dense conjugate direction field, both of which are used to extract the final PQ mesh. To train and validate our network, we generate a large synthetic dataset that mimics architect sketching of freeform quadrilateral patches. The effectiveness and usability of our system are demonstrated with quantitative and qualitative evaluation as well as user studies.

Index Terms—Freeform surface, architectural geometry, planar quadrilateral mesh, sketch-based modeling, deep learning

✦

1 INTRODUCTION

FREEFORM 3D architectural modeling usually starts with the concept design, which can be conveyed via 2D sketching. The sketching depicts the boundary and the profile of the surface, and directional strokes may be added to describe the panel layout (see Fig. 1 for two freeform roof designs via 2D sketching). Afterwards, the 2D sketch needs to be converted into a 3D digital model. To do so, architects usually need to use 3D modeling software to manually create surface geometry according to the sketch and spend a considerable amount of time editing it into the desired shape. The created geometry is then decomposed into panels, which often involves computational routines that enforce geometric constraints related to construction requirements and design intent [1]. This overall workflow can be time-consuming and requires professional knowledge at each step.

An alternative approach is sketch-based modeling [2] which automatically converts 2D sketches into 3D shapes. The main obstacle in this process is the ambiguity in the mapping from 2D to 3D which makes the problem ill-posed. In addition, the sketches can be inaccurate and noisy, which makes the conversion problem even more challenging. Over the past two decades, many approaches have been proposed to address these challenges. Geometry priors from shape classes [3], [4], [5], [6] provide a strong cue to resolve
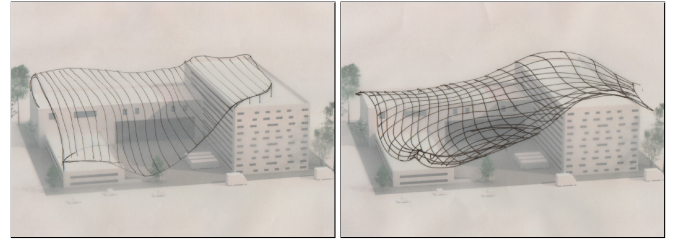

Fig. 1. Two different designs of freeform roofs via 2D sketching.

the ambiguity, but it is not easy to generalize them to unseen shape classes such as freeform shapes. Sketching from multiple views [7], [8] is a natural way to remove ambiguity and create a freeform design, but it could be inconvenient for the user to change the viewport frequently and provide more 2D sketches for conveying their rough design ideas. Moreover, panel layouts are an important part of freeform architectural design that induces geometric constraints, but they are rarely considered in existing approaches.

In this paper, we propose a sketch-based modeling system that allows users to design roof-like freeform architectural surfaces represented as planar quadrilateral (PQ) meshes. A PQ mesh represents a layout of flat panels with quadrilateral shapes, which is a preferable way to realize freeform glass structures due to their benefits in the construction cost [9]. To create such shapes, existing methods typically start from a reference surface that is modeled separately, and optimize a quadrilateral mesh to enforce face planarity while aligning with the reference shape [10], [11]. Our system enables a new modeling paradigm where the underlying surface shape and the PQ mesh layout are determined simultaneously from a single 2D sketch. To specify the surface structure, the user can draw the boundary lines and suggestive contour

- Z. Deng is with School of Data Science, University of Science and Technology of China.
- H. Pan and Y. Liu are with Internet Graphics Group, Microsoft Research Asia.
- W. Jabi is with the Welsh School of Architecture, Cardiff University.
- J. Zhang is with School of Mathematical Sciences, University of Science and Technology of China.
- B. Deng is with School of Computer Science and Informatics, Cardiff University.

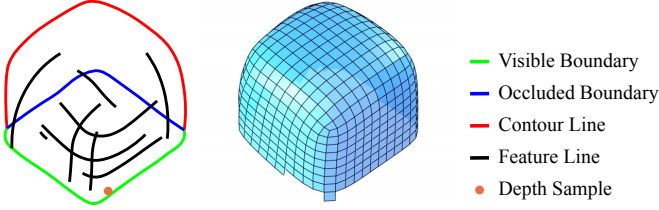[†]*Corresponding author. Email:* `DengB3@cardiff.ac.uk`.

Fig. 2. An example sketch using our system and the resulting PQ mesh.

lines, as well as annotate the self-occluded parts. In addition, the user can draw sparse feature lines within the surface area to indicate the edge directions of the PQ mesh. To determine the mesh at an interactive rate, we train a deep neural network that takes the 2D sketch as input and first infers the depth and normal maps of the visible occluded regions, which are further fused via a network module to a smooth B-spline surface patch that represents the underlying surface shape. The network also infers a dense conjugate direction field (CDF) over the surface as an indication of the PQ mesh layout [10]. The final PQ mesh is then extracted from the B-spline surface and the CDF. To train and test our neural network, we propose a method to synthesize a large amount of data containing roof-like B-spline surfaces, their PQ mesh layouts, and the corresponding sketches. We verify the effectiveness and usability of our system using a series of quantitative and qualitative evaluations, ablation studies, and user studies. To summarize, our contribution includes:

- We develop a novel sketch-based modeling system for PQ meshes. Utilizing deep neural networks, our system can produce a PQ mesh from a single sketch at an interactive rate, providing an intuitive way to model the surface shape and the PQ mesh layout simultaneously.
- We propose a method to generate a large dataset of surfaces and sketches that is suitable for the training and testing of our network.

2 RELATED WORK

In this section, we briefly review existing works closely related to our paper. More in-depth exposition of differential geometry and deep learning can be found in [12] and [13].

Sketch-based modeling. Sketch-based modeling has been studied extensively in the past. In the following, we focus on methods that create 3D shapes from 2D sketches without the need for an existing 3D model, as the same scenario is considered in this paper. The readers are referred to [2], [14] for more complete overviews that cover other scenarios.

Many existing works utilize geometry priors from human experience or specific object categories to determine the 3D shapes. Igarashi *et al.* [15] utilized regular shape composition to model stuffed animals and other rotund objects. Tai *et al.* [16] designed 3d models from sketched silhouette curves based on a convolution surface model. Schmidt *et al.* [17] increased the shape complexity by using Hierarchical Implicit Volume Models as an underlying shape representation and applying blending and CSG operations to implicit volumes inflated from 2D contours. To model 3D trees, Chen *et al.* [18] first constructed a 3D branch model using a database of trees as prior, and then added leaves based on botanical

rules. Chen *et al.* [4] constructed an architectural model from a freehand sketch by identifying primitive geometries, detailed geometries, and textures using databases of such elements as priors. Gingold *et al.* [5] developed a sketching system where the user can specify semantic information that helps to interpret the 2D sketch. Rivers *et al.* [19] modeled 3D objects from multi-view silhouettes by constructing 3D parts from the silhouettes and assembling them using CSG operations. Entem *et al.* [20] used smoothness and structural symmetry priors to model 3D animals from a side-view sketch. Jung *et al.* [21] inferred quasi-developable surfaces with pre-designed folds from multi-view sketches, by interleaving developability optimization with identification of silhouette points. Dvorožňák *et al.* [22] combined 3D inflation with a rigidity-preserving and layered deformation model to produce a smooth 3D mesh from a single-view sketch. Li *et al.* [23] utilized detailed curvature hints in the 2D sketch to infer local shapes and model freeform surfaces. Similarly, the sparse feature lines used in our system follow the PQ mesh layout and provide a strong hint for the local curvature.

Recently, learning-based approaches have become a promising way to handle different types of sketch inputs and predict 3D shapes robustly. Huang *et al.* [24] and Nishida *et al.* [25] utilized deep convolutional neural networks (CNN) to predict procedural modeling parameters from the input sketch. Han *et al.* [26] proposed a sketching system for 3D face and caricature modeling, using a CNN-based deep regression network to infer the coefficients for a bilinear face representation. Lun *et al.* [7] proposed an encoder-decoder network to infer multi-view depth and normal maps from the input sketch, and consolidated them into a point cloud via optimization. Su *et al.* [27] inferred normal maps from 2D sketches by treating the process as an image translation problem and solving it using a Generative Adversarial Network (GAN) framework. A GAN framework was also used in [28] to reconstruct 3D models from hand-drawn sketches. Delanoy *et al.* [8] proposed an encoder-decoder network to predict the occupancy of 3D voxels from a line drawing. To infer freeform surfaces from sparse sketches, Li *et al.* [29] used CNNs to infer the depth and normal maps representing the surface, with an intermediate layer that models the curvature direction field and produces a confidence map to improve robustness. In comparison, we also utilize neural networks to infer depth maps for the target surface, but with additional modules to infer conjugate direction fields that help to extract a PQ mesh. Du *et al.* [30] created animal-like 3D head meshes from dual-view sketches, using Graph Convolutional Neural Networks to initialize and refine the model. Yan *et al.* [31] adopted a conditional generative adversarial network (CGAN) trained with physics-based simulation data to produce raw liquid splash models from input sketches, and further refined them to achieve realistic results. Han *et al.* [32] reconstructed a 3D model from multiple sketches by optimizing the 3D shape to match a set of attenuance images generated from the sketches using a CGAN. Li *et al.* [33] presented a CAD modeling system where a deep neural network infers shape edits from the user sketches to incremental create a 3D object. Yang *et al.* [34] proposed a two-stage learning framework for 3D face reconstruction from sketches, integrating the knowledge of face reconstruction from photos. To generate 3D shapes
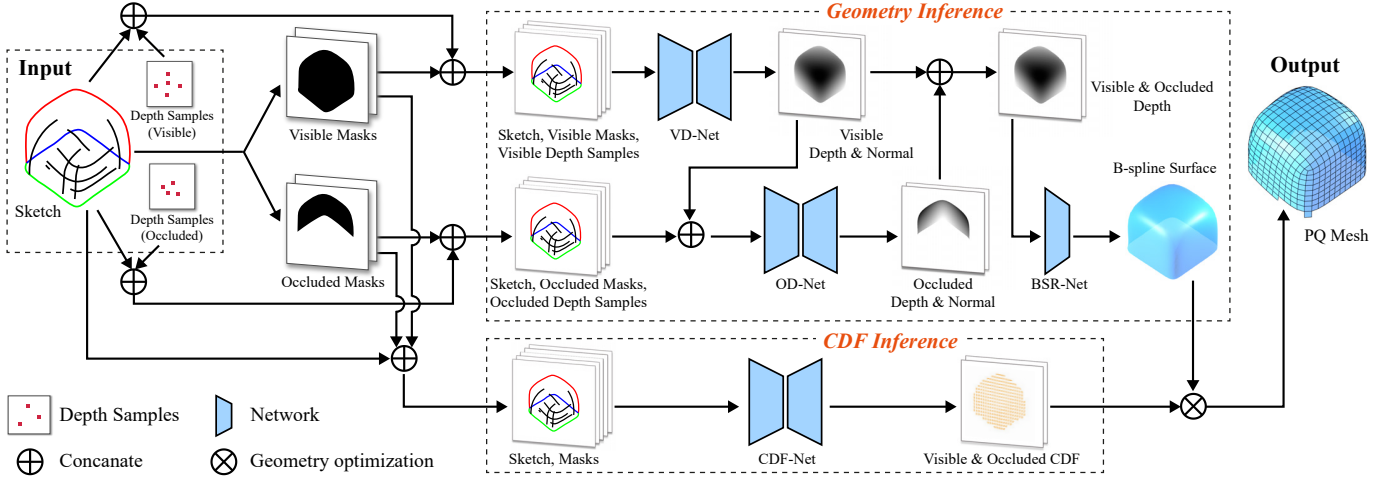
Fig. 3. The algorithm pipeline of our Sketch2PQ system. The system takes stroke lines, depth samples, and the visible and occluded region masks induced from the sketch as input. A geometry inference module predicts the depth and normal maps for the visible and occluded regions, and use them to infer a B-spline surface. In addition, a conjugate direction field (CDF) inference module predicts a CDF that approximates the PQ mesh layout. Finally, a PQ mesh is extracted from the B-spline surface and the CDF via geometry optimization.

from poorly-drawn sketches, Zhang *et al.* [35] proposed an encoder-decoder architecture that disentangles the shape and the viewpoint in the latent space. Smirnov *et al.* [36] used a deformable template consisting of Coons patches to learn 3D man-made shapes from 2D sketches.

PQ meshes. Discretization of freeform surfaces using PQ meshes has been studied from both theoretical and practical perspectives. Liu *et al.* [10] noted that PQ meshes are discrete counterparts of conjugate curve networks; they derived special PQ meshes including conical and circular meshes from networks of principal curvature lines, a special type of conjugate curve network. Pottmann *et al.* [37] proposed methods to compute parallel meshes with planar faces, with applications for multiple-layer freeform structures. Zadravec *et al.* [38] and Liu *et al.* [11] designed general PQ meshes by computing smooth conjugate direction fields (CDF) on the reference surface. The CDF computation can also be done by optimizing an N-PolyVector field [39] without the need for integer variables. All the methods above seek a PQ mesh that approximates an existing surface. Another way of PQ mesh design is to explore the shape space of PQ meshes with the same mesh topology but different vertex positions [40], [41], [42], [43], [44], [45], [46], starting from an existing PQ shape. Neither approach is suitable for sketch-based modeling of PQ meshes, since they both require an initial 3D shape which is not available from the user input.

3 THE SKETCH2PQ MODELING SYSTEM

We develop a sketch-based modeling system for freeform PQ structures (see Fig. 2 for its pipeline). The system supports four types of strokes for depicting the surface structure and the PQ mesh layout (Sec. 3.1). The sketch image is fed to a neural network to infer a B-spline surface for the underlying shape, as well as a conjugate direction field (CDF) that indicates the PQ mesh layout. The final PQ mesh is generated from the surface shape and the CDF (Sec. 3.2).
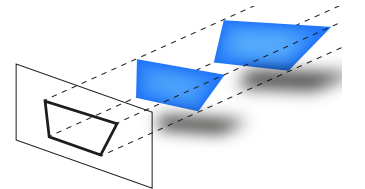
3.1 System Input

Stroke types. The input to our system is an architectural style sketch represented as an RGB image, depicting the axonometric projection of a roof-like surface with a boundary. Following the design habit of architectural users, we allow two classes of stroke lines (see Fig. 2 for an example):

- *Structural lines* convey the structural information of the surface. There are three types of structural lines:
  - A *visible boundary* is a visible part of the surface boundary; it is shown in **green**.
  - An *occluded boundary* is an occluded part of the surface boundary; it is shown in **blue**.
  - A *contour line* is part of the boundary of the surface area on the canvas but is not part of the 3D surface boundary; it is shown in **red**.
- *Feature lines* are the direction lines that the resulting PQ mesh edges should follow; they are shown in **black**.

Similar to [23], after the user draws a stroke line on the 2D canvas, it is vectorized to a cubic B-spline curve via curve approximation for easy editing. To reduce ambiguity, we assume that the features lines are drawn on visible surface regions only. The vectorized sparse sketch is rendered on a 256×256 RGB image with a line width of three pixels, and we apply a 3×3 Gaussian filter on the original image.

The stroke lines roughly follow the boundaries, contours, and a subset of the interior edges of a PQ mesh. This can only determine the PQ mesh shape up to scaling and translation along the view direction. This is because for any quadrilateral with co-planar vertices in the 3D space, scaling and/or translation along the view direction will keep the vertices co-planar while producing the same projection onto the viewing plane (see the inset figure for an example). As a result, there is a family of PQ meshes with the same projection onto the viewing plane. Our system only returns one PQ mesh among
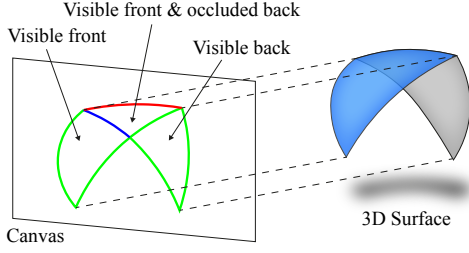
Fig. 4. An example of visibility deduced from a sketch.

Fig. 5. Examples of visibility-orientation masks for different sketches. Front and back masks of the same visibility are shown together.

the valid solutions. To enable more fine-grained control, we allow depth samples as optional inputs to provide depth cues at certain parts of the surface, similar to [29]. As each depth sample can be specified for either a visible or occluded part of the surface, we store two depth sample maps for the visible parts and the occluded parts, respectively. Each is represented as a single-channel map, which contains the specified depth values for pixels corresponding to the sample points and zero values for other pixels (note that we assume the whole 3D surface have positive depth values).

Visibility and orientation masks. The input sketch is first processed to derive visibility attributes that facilitate the inference of the surface shape. The attributes are fed into the network together with the user input. We consider the sketch as a union of disjoint regions (see Fig. 4), where each region is enclosed by structural lines (i.e., visible/occluded boundaries and contour lines) and contains no other structural lines in the interior. We call such a region a *minimal region*. All minimal regions can be identified using a seeded region growing algorithm [47]. For each minimal region, we determine: (1) whether it corresponds to a visible or occluded part of the surface, and (2) the orientation of the corresponding surface part, i.e., whether its side facing the canvas is the front side or the back side of the surface. To determine the properties, we make the following assumptions:

Assumption 1. *The surface and the sketch satisfy:*

(a) The surface is of disk topology, and has no self-intersection.

(b) The surface projects onto a disk-topology region in the canvas.

(c) Any straight line parallel to the view direction has no more than two intersection points with the surface.

(d) For a minimal region that corresponds to an occluded part of the surface, its enclosing structural lines must contain a contour line.

(e) Two structural lines can only overlap at isolated points.

(f) Among all visible parts of the surface, those with their front side facing the canvas have a larger total projected area on the canvas than those with their back side facing the canvas.

These assumptions help to simplify the problem while covering a sufficiently large space of surface shapes. Under the assumptions, each minimal region corresponds to a visible part of the surface. We determine the orientation of all visible parts based on the adjacency relation between their corresponding minimal regions and the following properties:

Proposition 1. *The orientations of the visible parts satisfy:*

(a) If two minimal regions share a visible boundary line, then their corresponding visible parts have opposite orientations.
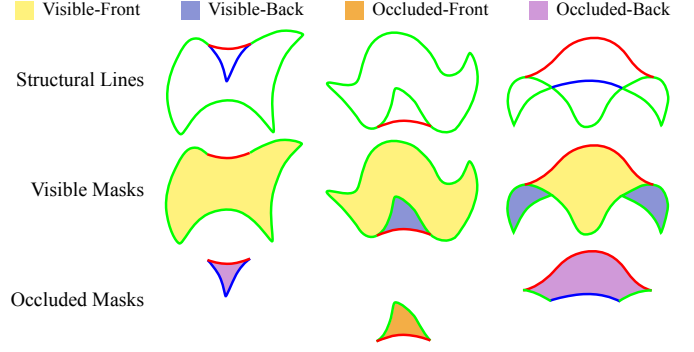
(b) If two minimal regions share an occluded boundary line, then their corresponding visible parts have the same orientation.

A proof is given in Appendix A. Accordingly, we first pick an arbitrary minimal region and set the orientation of its corresponding visible part to be front, then propagate the orientation information to all visible parts using Proposition 1. We then compute the total areas of the minimal regions with front and back orientations, respectively. If the back regions have a larger total area, we negate the orientation of all visible parts according to Assumption 1(f). Finally, Assumption 1(d) implies that each minimal region incident with a contour line corresponds to a visible part and an occluded part with opposite orientations, and we use this rule to identify all occluded parts and their orientations.

We divide all surface parts into four categories based on their visibility and orientation: *visible-front*, *visible-back*, *occluded-front*, and *occluded-back*. For each category, we construct a binary mask indicating all minimal regions that correspond to surface parts of the category (see Fig. 5 for some examples). These masks are fed to the neural networks along with the input sketch and depth sample maps. They are also used to post-process the output of the networks, by discarding information that is outside the surface region.

3.2 Network Structure and Loss Functions

The user input and the visibility-orientation masks obtained in Sec. 3.1 are used to derive a PQ mesh. As the problem is highly non-linear, direct inference with a single network often produces unsatisfactory results. We instead utilize multiple modules to infer different shape attributes that determine the final PQ mesh. Specifically, we first use two network modules to predict the depth and normal maps for the visible and occluded parts of the surface, respectively (Secs. 3.2.1 & 3.2.2). The depths are fed to another network module to infer the underlying surface shape represented as a B-spline surface (Sec. 3.2.3). In addition, a network module predicts a conjugate direction field (CDF) that serves as an indication of the PQ mesh layout [10] (Sec. 3.2.4). Finally, a geometry optimization is performed to compute the final PQ mesh according to the B-spline surface and the CDF (Sec. 3.2.5).

3.2.1 Depth and normal prediction for visible parts

Given the input sketch, the visible-front and visible-back masks, and the visible depth samples, we adopt a U-Net

like network module called *VD-Net* (see Fig. 6) to recover the depth and outward normal maps over the 2D canvas domain for the visible surface parts, similar to [29]. Its input contains six channels: three for the RGB sketch image, two for the visible-front and visible-back masks, and one for the (optional) depth samples. Our VD-Net has five domain resolutions, from 256×256 to 16×16. The output is in eight channels: two for the depth maps of the visible-front and visible-back surface parts, and six for their normal maps. The network is trained using the following loss function:

$$E_{\mathrm{v}} = E_{\mathrm{v,dat}} + w_{\mathrm{v,smo}} E_{\mathrm{v,smo}} + w_{\mathrm{v,comp}} E_{\mathrm{v,comp}} + w_{\mathrm{v,dep}} E_{\mathrm{v,dep}}, \quad (1)$$

where the terms $E_{\mathrm{v,dat}}, E_{\mathrm{v,smo}}, E_{\mathrm{v,comp}}, E_{\mathrm{v,dep}}$ enforce different properties as explained below. Their weights are empirically chosen as $w_{\mathrm{v,smo}} = 0.2$, $w_{\mathrm{v,comp}} = 0.1$ and $w_{\mathrm{v,dep}} = 10$.

Data term. The term $E_{\mathrm{v,dat}}$ penalizes the L_2 error between the predicted depth/normal maps and their ground-truth counterparts on the visible surface parts:

$$E_{\mathrm{v,dat}} = \frac{1}{|\mathcal{P}_{\mathrm{v}}|} \sum_{i \in \mathcal{P}_{\mathrm{v}}} \left(\sigma_{\mathrm{d}} (d_i - d_i^{\mathrm{gt}})^2 + \sigma_{\mathrm{n}} \|\mathbf{n}_i - \mathbf{n}_i^{\mathrm{gt}}\|^2 \right), \quad (2)$$

where $\mathcal{P}_{\mathrm{v}}$ denotes the set of pixels for the visible surface parts, and $d_i, d_i^{\mathrm{gt}} \in \mathbb{R}$, $\mathbf{n}_i, \mathbf{n}_i^{\mathrm{gt}} \in \mathbb{R}^3$ are the predicted and ground-truth depth and normal at a pixel i, respectively. The weights $\sigma_{\mathrm{d}}, \sigma_{\mathrm{n}}$ are set to $\sigma_{\mathrm{d}} = 1.0, \sigma_{\mathrm{n}} = 0.1$.

Smoothness term. $E_{\mathrm{v,smo}}$ is a smoothness regularization term for the depth and normal maps, which penalizes the difference between the values in neighboring pixels:

$$E_{\mathrm{v,smo}} = \frac{1}{|\mathcal{N}_{\mathrm{v}}|} \sum_{(i,j) \in \mathcal{N}_{\mathrm{v}}} \left(\tau_{\mathrm{d}} (d_i - d_j)^2 + \tau_{\mathrm{n}} \|\mathbf{n}_i - \mathbf{n}_j\|^2 \right). \quad (3)$$

Here $\mathcal{N}_{\mathrm{v}}$ is the set of neighboring visible pixels with the same orientation, and the weights are set to $\tau_{\mathrm{d}} = 0.01, \tau_{\mathrm{n}} = 0.001$.

Depth-normal compatibility term. The predicted normal vectors and depth values should be compatible with each other. This is enforced via the following term:

$$E_{\mathrm{v,comp}} = \frac{1}{|\mathcal{R}_{\mathrm{v}}|} \sum_{i \in \mathcal{R}_{\mathrm{v}}} (\mathbf{n}_i \cdot \mathbf{t}_{i,x})^2 + \frac{1}{|\mathcal{U}_{\mathrm{v}}|} \sum_{i \in \mathcal{U}_{\mathrm{v}}} (\mathbf{n}_i \cdot \mathbf{t}_{i,y})^2,$$

where $\mathcal{R}_{\mathrm{v}}$ is the set of visible pixels where their right neighbor pixels are also visible with the same orientation, i.e., $\mathcal{R}_{\mathrm{v}} = \{i \mid (i, r(i)) \in \mathcal{N}_{\mathrm{v}}\}$ where $r(\cdot)$ denotes the right neighbor pixel. Similarly, $\mathcal{U}_{\mathrm{v}} = \{i \mid (i, u(i)) \in \mathcal{N}_{\mathrm{v}}\}$ where $u(\cdot)$ denotes the upper neighbor pixel. $\mathbf{t}_{i,x} = (1, 0, (d_{r(i)} - d_i)/\delta)$, $\mathbf{t}_{i,y} = (1, 0, (d_{u(i)} - d_i)/\delta)$ are tangent vectors at pixel i computed by finite difference, where δ is the pixel width when mapped to the canonical scale of the training data. $E_{\mathrm{v,comp}}$ enforces orthogonality between the predicted normals and the tangent vectors derived from the predicted depths.

Depth sample term. Finally, $\mathcal{S}_{\mathrm{v}}$ requires the predicted depth to be close to the provided depth samples:

$$E_{\mathrm{v,dep}} = \begin{cases} \frac{1}{|\mathcal{S}_{\mathrm{v}}|} \sum_{i \in \mathcal{S}_{\mathrm{v}}} (d_i - d_i^{\mathrm{s}})^2, & \text{if } \mathcal{S}_{\mathrm{v}} \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

where $\mathcal{S}_{\mathrm{v}}$ is the set of visible pixels with depth samples, and d_i^{s} is the provided depth value for pixel i.
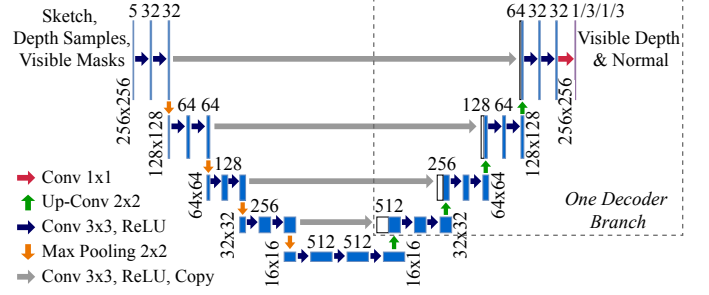

Fig. 6. The VD-Net is an encoder-decoder network with five domain resolutions. The input is a 6-channel image containing the sketch, the visible-front/back masks and the visible depth samples. Four decoder branches share the same encoder and predict the depth & normal maps for the visible front and back regions, respectively.

3.2.2 Depth and normal prediction for occluded parts

For the occluded parts, a setup similar to Sec. 3.2.1 may be used to infer the depth and outward normal maps. However, the results may be inconsistent with their counterparts for the visible regions, i.e., the depth and normal functions may be discontinuous across the contour lines. Therefore, we adapt the network structure and loss function to enforce the consistency. Specifically, we introduce a network structure, called *OD-Net*, with its input being the sketch, the occluded-front and occluded-back masks, the occluded depth samples, and the depth and normal maps produced by the VD-Net. The remaining parts of the network are the same as the VD-Net. We train the network using the following loss function:

$$\begin{aligned} E_{\mathrm{o}} = & E_{\mathrm{o,dat}} + w_{\mathrm{o,smo}} E_{\mathrm{o,smo}} + w_{\mathrm{o,comp}} E_{\mathrm{o,comp}} \\ & + w_{\mathrm{o,dep}} E_{\mathrm{o,dep}} + w_{\mathrm{cons}} E_{\mathrm{cons}}. \end{aligned} \quad (5)$$

Here $E_{\mathrm{o,dat}}, E_{\mathrm{o,smo}}, E_{\mathrm{o,comp}}, E_{\mathrm{o,dep}}$ are the data term, smoothness term, depth-normal compatibility term, and depth sample term for the occluded parts respectively, and are defined for the set $\mathcal{P}_{\mathrm{o}}$ of occluded pixels in a way similar to Sec. 3.2.1. E_{cons} is a geometric consistency term between the visible and occluded regions near the contour line:

$$\begin{aligned} E_{\mathrm{cons}} = & \frac{1}{|\Omega|} \sum_{(i,j) \in \Omega} \left[\gamma_{\mathrm{d}} ((d_i^{\mathrm{v}} - d_j^{\mathrm{o}})^2 + (d_i^{\mathrm{o}} - d_j^{\mathrm{v}})^2) \right. \\ & \left. + \gamma_{\mathrm{n}} (\|\mathbf{n}_i^{\mathrm{v}} - \mathbf{n}_j^{\mathrm{o}}\|^2 + \|\mathbf{n}_i^{\mathrm{o}} - \mathbf{n}_j^{\mathrm{v}}\|^2) \right]. \end{aligned} \quad (6)$$

Here $\Omega = \{(i,j) \mid i, j \in \mathcal{C} \cap \mathcal{P}_{\mathrm{v}} \cap \mathcal{P}_{\mathrm{o}}, j \in N_4(i)\}$, where $\mathcal{C}$ is the set of pixels that are no more than four pixels away from the contour line, and $N_4(i)$ denotes a neighborhood of pixel i which consists of i itself as well as its upper neighbor, right neighbor, and upper-right neighbor. The symbols $d^{\mathrm{v}}, \mathbf{n}^{\mathrm{v}}$ and $d^{\mathrm{o}}, \mathbf{n}^{\mathrm{o}}$ denote the depth and normal for the visible and occluded regions, respectively. The weights in Eq. (6) are set to $\gamma_{\mathrm{d}} = 1, \gamma_{\mathrm{n}} = 0.1$, and the weights in Eq. (5) are set to $w_{\mathrm{o,smo}} = 0.2, w_{\mathrm{o,comp}} = 0.1, w_{\mathrm{o,dep}} = 10, w_{\mathrm{cons}} = 0.001$.

3.2.3 Surface prediction and refinement

With the predicted depth maps for the visible and occluded parts, we reconstruct a cubic B-spline surface that represents the underlying shape of the PQ mesh. We use a B-spline representation because its smooth shapes are suitable for freeform architectural design. Moreover, it is compatible with NURBS-based 3D modeling software such as Rhino3D,

which is popular among architects. This allows the design to be easily integrated into existing architectural design workflows and further modified in a later stage. Therefore, even though it may be possible to infer the PQ mesh without the intermediate B-spline shape, we choose to reconstruct a B-spline surface to enable interoperability with the existing design pipeline. We use 30×30 B-spline control points, which are sufficient for representing a large variety of shapes. To determine the surface, one possibility is to optimize the control points to fit the depth maps. However, this is a non-convex problem due to the unknown correspondence between the depth map and the B-spline surface, and a good fitting would require proper initialization which is not a trivial problem. We instead use a network module, called *BSR-Net*, to predict the control points $\{\mathbf{p}_{i,j} \in \mathbb{R}^3 \mid 1 \leq i, j \leq 30\}$ from the depth maps produced by VD-Net and OD-Net. We choose ResNet-18 [48] as its structure and train it using a loss function:

$$E_{\text{surf}} = E_{\text{surf,dat}} + w_{\text{surf,fair}} E_{\text{surf,fair}}. \tag{7}$$

Here $E_{\text{surf,dat}}$ is a data term that measures the difference between the predicted and ground-truth B-spline surfaces:

$$E_{\text{surf,dat}} = \frac{1}{|\mathcal{S}_{\text{bsr}}|} \sum_{(u,v) \in \mathcal{S}_{\text{bsr}}} \|\mathbf{s}(u,v) - \mathbf{s}_{\text{gt}}(u,v)\|^2, \tag{8}$$

where $\mathcal{S}_{\text{bsr}}$ is a set of 100×100 parameters sampled regularly from the parameter domain of the surface, and $\mathbf{s}(u,v)$, $\mathbf{s}_{\text{gt}}(u,v)$ are the surface points at the parameter (u,v) using the predicted and ground-truth control points, respectively. $E_{\text{surf,fair}}$ is a fairness term for the predicted control net:

$$E_{\text{surf,fair}} = \sum_{i=2}^{29} \sum_{j=2}^{29} \|\mathbf{l}_{i,j}^u\|^2 + \|\mathbf{l}_{i,j}^v\|^2, \tag{9}$$

where $\mathbf{l}_{i,j}^u$ and $\mathbf{l}_{i,j}^v$ are the polyline Laplacians for the two control polygons at $\mathbf{p}_{i,j}$:

$$\mathbf{l}_{i,j}^u = \mathbf{p}_{i,j} - \frac{\mathbf{p}_{i-1,j} + \mathbf{p}_{i+1,j}}{2}, \quad \mathbf{l}_{i,j}^v = \mathbf{p}_{i,j} - \frac{\mathbf{p}_{i,j-1} + \mathbf{p}_{i,j+1}}{2}.$$

The weight $w_{\text{surf,fair}}$ is set to 0.1. During the test phase, we use the predicted surface as initialization and optimize its control points to further align it with the depth maps:

$$\min_{\{\mathbf{p}_{i,j}\}} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{q}_i \in \mathcal{D}} (\text{dist}(\mathbf{q}_i))^2 + w_{\text{f}} E_{\text{surf,fair}}, \tag{10}$$

where $\mathcal{D}$ is the set of target 3D points derived from the input depth maps, $\text{dist}(\mathbf{q}_i)$ is the distance from a point $\mathbf{q}_i$ to the surface, and $w_{\text{f}} = 2$. The problem is solved using an iterative B-spline fitting method [49]. As the initial surface is close to the target points, typically a few iterations are sufficient.

3.2.4 Conjugate direction field prediction

The feature lines in the sketch indicate the edge directions of the PQ mesh layout. To determine the PQ mesh, we need to first compute a dense conjugate direction field (CDF) on the surface that aligns with the feature lines. A CDF is a general cross field consisting of four directions for each point, which correspond to the edges of infinitesimally small planar quadrilaterals [10]. As such, it is a suitable approximation of a PQ mesh layout and commonly used as initialization for PQ mesh generation [11]. Existing methods for CDF computation need to solve a non-linear and non-convex optimization problem with a large number of variables [11], [39], which
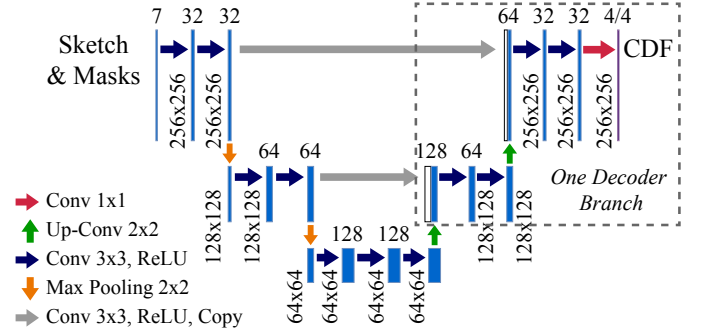

Fig. 7. The CDF-Net is an encoder-decoder network with three domain resolutions. The input is a 7-channel image for the sketch and its four visibility-orientation masks. Two decoder branches share the same encoder and predict the CDF for the visible and occluded regions.

is too slow for our interactive system. Therefore, we design a network module, called *CDF-Net*, to efficiently predict the CDF. Since the CDF directions are 3D vectors in the tangent plane at each point of the surface, it is sufficient to predict their 2D projections onto the canvas plane. The predicted 2D vectors can then be projected back into the corresponding tangent planes on the B-spline surface to obtain the 3D CDF directions. To this end, our CDF-Net predicts a dense CDF for the visible regions and the occluded regions respectively.

The CDF-Net has a U-Net structure (see Fig. 7), with a 7-channel input that includes the RGB sketch image and its four visibility-orientation masks. The output is a dense map with four channels, with a value $(c_{j,0}, c_{j,1}, c_{j,2}, c_{j,3})$ at each pixel j to represent two vectors $\mathbf{u}_j = (c_{j,0}, c_{j,1})$ and $\mathbf{v}_j = (c_{j,2}, c_{j,3})$ that form the predicted projected CDF $\{\mathbf{u}_j, -\mathbf{u}_j, \mathbf{v}_j, -\mathbf{v}_j\}$. The network is trained to produce a prediction that is close to the ground-truth vectors $\{\mathbf{a}_j, -\mathbf{a}_j, \mathbf{b}_j, -\mathbf{b}_j\}$ at each pixel j. However, since each quadruplet of vectors is treated as an un-ordered set, there is an ambiguity in their correspondence when evaluating the difference between the predicted and ground-truth quadruplets. A similar ambiguity arises if we compare the predicted quadruplets at two neighboring pixels to evaluate the smoothness of the CDF. To avoid such issues, we follow [29] and encode each quadruplet of vectors using their 4-PolyVector representation [39]. Specifically, we treat the predicted vectors as complex numbers, i.e., $\mathbf{u}_j' = c_{j,0} + ic_{j,1}, \mathbf{v}_j' = c_{j,2} + ic_{j,3}$. Then the quadruplet $\{\mathbf{u}_j', -\mathbf{u}_j', \mathbf{v}_j', -\mathbf{v}_j'\}$ are the roots of a complex polynomial $P(z) = (z - \mathbf{u}_j')(z + \mathbf{u}_j')(z - \mathbf{v}_j')(z + \mathbf{v}_j') = z^4 - (\mathbf{u}_j'^2 + \mathbf{v}_j'^2)z^2 + \mathbf{u}_j'^2 \mathbf{v}_j'^2$. We then identify the triplet with the polynomial coefficients $\mathbf{u}_j'^2 + \mathbf{v}_j'^2 = \hat{c}_{j,0} + i\hat{c}_{j,1}$ and $\mathbf{u}_j'^2 \mathbf{v}_j'^2 = \hat{c}_{j,2} + i\hat{c}_{j,3}$, and encode them as $\hat{\mathbf{c}}_j = (\hat{c}_{j,0}, \hat{c}_{j,1}, \hat{c}_{j,2}, \hat{c}_{j,3})$. The ground-truth vectors are encoded in the same way. We then evaluate the difference between two quadruplets by comparing their encoding vectors in $\mathbb{R}^4$. Based on this encoding scheme, we train the CDF-Net with the following loss function:

$$E_{\text{cdf}} = E_{\text{cdf,dat}} + w_{\text{cdf,smo}} E_{\text{cdf,smo}} + w_{\text{cdf,cons}} E_{\text{cdf,cons}}. \tag{11}$$

Here $E_{\text{cdf,dat}}$ is a data term that penalizes the difference between the predicted CDF and the ground-truth:

$$E_{\text{cdf,dat}} = \frac{1}{|\mathcal{P}_{\text{v}}|} \sum_{j \in \mathcal{P}_{\text{v}}} \|\hat{\mathbf{c}}_j^{\text{v}} - \hat{\mathbf{c}}_j^{\text{v,gt}}\|^2 + \frac{1}{|\mathcal{P}_{\text{o}}|} \sum_{j \in \mathcal{P}_{\text{o}}} \|\hat{\mathbf{c}}_j^{\text{o}} - \hat{\mathbf{c}}_j^{\text{o,gt}}\|^2$$
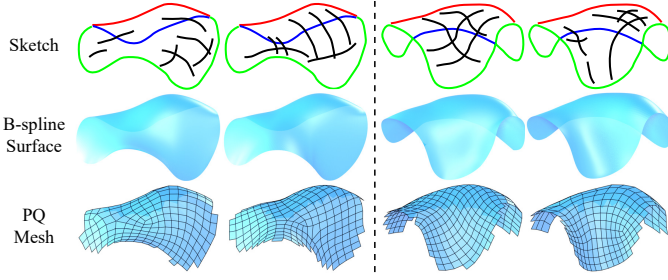
Fig. 8. Feature lines can control the PQ mesh layout. Here we show two pairs of sketches, each having the same structural lines but different feature lines. The PQ mesh edges align well with the feature lines.

where $\hat{\mathbf{c}}_j^{\mathrm{v}}$ and $\hat{\mathbf{c}}_j^{\mathrm{o}}$ are the CDF encoding vectors at pixel j for the visible and occluded region respectively, and $\hat{\mathbf{c}}_j^{\mathrm{v,gt}}, \hat{\mathbf{c}}_j^{\mathrm{o,gt}}$ are the ground-truth encoding vectors. The term $E_{\mathrm{cdf,smo}}$ evaluates the smoothness of the predicted CDF by comparing its encoding vectors at neighboring pixels:

$$E_{\mathrm{cdf,smo}} = \frac{1}{|\mathcal{N}_{\mathrm{v}}|} \sum_{(j,k)\in\mathcal{N}_{\mathrm{v}}} \|\hat{\mathbf{c}}_j^{\mathrm{v}} - \hat{\mathbf{c}}_k^{\mathrm{v}}\|^2 + \frac{1}{|\mathcal{N}_{\mathrm{o}}|} \sum_{(j,k)\in\mathcal{N}_{\mathrm{o}}} \|\hat{\mathbf{c}}_j^{\mathrm{o}} - \hat{\mathbf{c}}_k^{\mathrm{o}}\|^2,$$

where $\mathcal{N}_{\mathrm{v}}$ and $\mathcal{N}_{\mathrm{o}}$ are the sets of neighboring pixel pairs in the visible and occluded regions, respectively. The term $E_{\mathrm{cdf,cons}}$ enforces consistency of the CDF across the contour line, similar to the term E_{cons} as in (6):

$$E_{\mathrm{cdf,cons}} = \frac{1}{|\Omega|} \sum_{(j,k)\in\Omega} (\|\hat{\mathbf{c}}_j^{\mathrm{o}} - \hat{\mathbf{c}}_k^{\mathrm{v}}\|^2 + \|\hat{\mathbf{c}}_j^{\mathrm{v}} - \hat{\mathbf{c}}_k^{\mathrm{o}}\|^2), \quad (12)$$

where the set Ω is the same as in Eq. (6). The weights in Eq. (11) are set to $w_{\mathrm{cdf,smo}} = 0.1$ and $w_{\mathrm{cdf,cons}} = 0.01$.

3.2.5 PQ mesh generation

Once the projected CDF and the B-spline surface patch are obtained, we are ready to generate a planar quadrilateral mesh that follows the CDF as much as possible. We first discretize the B-spline surface into a triangle mesh according to pixel resolution via depth buffer rasterization, and project the CDF along the view direction onto the mesh. Then we utilize a field-based quad meshing algorithm [11] to extract a quadrilateral mesh that is approximately planar, and further optimize the face planarity using the method from [50]. Since the CDF already provides a good approximation of the edge directions in a PQ mesh, the final PQ mesh typically aligns well with the CDF and the sketched feature lines (see Fig. 8).

4 DATA GENERATION AND NETWORK TRAINING

The training data of our network needs to contain the ground-truth B-spline surfaces, their CDFs, and compatible sketch images. To the best of our knowledge, there is no large-scale 3D model dataset for freeform architectural shapes. Therefore, we build our own dataset for training and testing.

Shape creation. Since our system is aimed at roof-like structures, we assume the sketched shape to be a height-field surface $z = f(x, y)$. Therefore, we create each ground-truth surface by first randomly generating a height-field B-Spline surface, and then applying a random 3D deformation to introduce variations. Specifically, we generate the control points $\{\mathbf{p}_{i,j} \, (1 \leq i, j \leq 30)\}$ in the following steps.

First, we generate the (x, y)-coordinates $\overline{\mathbf{p}}_{i,j}$ for each control point. We first compute the candidate coordinates for the boundary control points (i.e., where at least one of i and j is in $\{1, 30\}$) using polar coordinates as $\widetilde{\mathbf{p}}_{i,j} = (\rho_{i,j}\cos\theta_{i,j}, \rho_{i,j}\sin\theta_{i,j})$, where the radial parameters $\{\rho_{i,j}\}$ are randomly sampled from $[2, 10]$, and the angle parameters $\{\theta_{i,j}\}$ are randomly sampled from $[0, 2\pi)$ and sorted in the same order as the control points along the boundary. We connect these 2D points according to the boundary connectivity of their corresponding control points to form a boundary polygon (see inset). To achieve a height field surface, we would like the (x, y)-coordinates of the remaining control points to be inside this polygon. To this end, we evenly sample 10000 points $\{\mathbf{a}_k\}$ inside the polygon, and optimize all $\overline{\mathbf{p}}_{i,j}$:

$$\min_{\{\overline{\mathbf{p}}_{i,j}\}} \quad \sum_{(i,j)\in\mathcal{B}} \|\overline{\mathbf{p}}_{i,j} - \widetilde{\mathbf{p}}_{i,j}\|_2^2 + \lambda_1 \sum_k \|\mathbf{a}_k - \overline{\mathbf{p}}_{\rho_k}\|_2^2$$
$$+ \lambda_2 \sum_{(i,j)\in\mathcal{I}} \|\overline{\mathbf{p}}_{i,j} - \mathbf{a}_{\eta_{i,j}}\|_2^2 + \lambda_3 E_{\mathrm{fair}}.$$

Here $\mathcal{B}$ and $\mathcal{I}$ are the index sets of the boundary and non-boundary control points, respectively. The first term above requires the boundary coordinates to be close to their candidate values. The second term penalizes the distance from each sample point $\mathbf{a}_k$ to the (x, y)-coordinates of its closest non-boundary control point $\mathbf{p}_{\rho_k}$. The third term aligns the non-boundary control point coordinates $\overline{\mathbf{p}}_{i,j}$ to its closest sample point $\mathbf{a}_{\eta_{i,j}}$. E_{fair} is a fairness term for the coordinates $\{\overline{\mathbf{p}}_{i,j}\}$, similar to $E_{\mathrm{surf,fair}}$ in Eq. (9). The weights are chosen as $\lambda_1 = \lambda_2 = 1$ and $\lambda_3 = 10$. We apply a global translation and scaling to all the optimized (x, y)-coordinates such that their bounding box is within $[-4, 4]^2$.

Afterwards, we generate the z-coordinates of the control points, to achieve a smooth appearance consistent with the style of freeform architecture. Let $\mathbf{z} \in \mathbb{R}^{900}$ be a vector that concatenates all the z-coordinates. We construct a matrix $\mathbf{L}$ such that the value $E_f(\mathbf{z}) = \mathbf{z}^T \mathbf{L}^T \mathbf{L} \mathbf{z}$ is a Laplacian fairness measure of the z-coordinates similar to the term $E_{\mathrm{surf,fair}}$ in Eq. (9). The desirable z-coordinates should achieve a small value of $E_f(\mathbf{z})$. Therefore, we perform eigendecomposition on the matrix $\mathbf{L}^T \mathbf{L}$ and select the 30 eigenvectors with the smallest eigenvalues. We then linearly combine these vectors using random coefficients from $[-1, 1]$, and scale the resulting vector such that all its components are in $[-5, 5]$. These components are used as the z-coordinates.

Finally, we transform the control points with a random freeform deformation (FFD) [51], which is defined using tricubic B-spline bases with $16\times16\times16$ uniform control points over the domain $[-5, 5]^3$. To induce the deformation, the x- and y-coordinates of each FFD control point are displaced by a random 2D vector from $[-0.2, 0.2]^2$. Furthermore, to ensure smoothness of the final result, we only retain the surface if the maximum magnitude of its Gaussian curvature is no larger than a threshold (chosen to be 5 in our experiments).

The above steps can generate shapes with a wide range of variations, but may not provide good coverage of regular shapes such as ellipsoidal patches. Therefore, we further
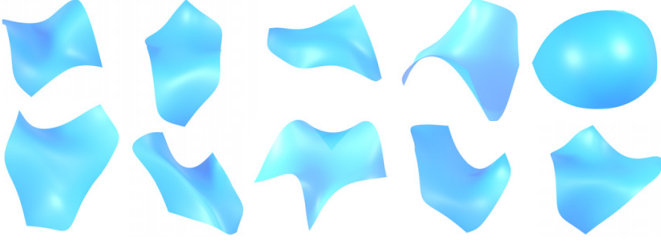
Fig. 9. Examples of B-spline surfaces generated using the method described in Sec. 4.

Fig. 10. The GUI of our system. The drawing canvas is on the left, and the modeling result is updated in real-time on the right.

extract height field patches from random ellipsoids and random cuboids by cutting the shapes with random planes, and fit B-spline surfaces to such patches as part of our dataset. Fig. 9 shows some examples of B-spline surfaces from our dataset. More examples are shown in Appendix E.

Sketch generation. To generate an axonometric view of a surface, we first place the camera at $(R/\sqrt{2}, R/\sqrt{2}, R)$ and point it towards the origin, with R being a sufficiently large positive value so that the whole surface is contained in the view volume. Then we rotate the surface with respect to the z-axis by an angle $k \cdot \pi/30$ ($k = 1, \ldots, 60$) to produce 60 rotated surfaces, and perform orthographic projection on each of them onto the viewing plane to generate the sketched structural lines (i.e., the visible/invisible boundaries and the contour lines) and the visibility/orientation masks. The visibility is determined using the Z-buffer, and the contour lines are determined using the method from [52]. We only retain the sketches that satisfy Assumption 1. For these sketches, we further generate features lines that indicate the PQ mesh layout. To this end, we first specify random conjugate directions at a few random locations on the triangulated surface, and compute a matching dense CDF using the method of [39]. A quad mesh that follows the CDF is then extracted using the method from [11]. Afterwards, we randomly select and trace some mesh edges from the visible region and project them to the viewing plane to obtain the feature lines. All the structural lines and feature lines are rendered as mentioned in Sec. 3.1. We also randomly select up to six depth samples. Moreover, to achieve more effective learning, we relabel the B-spline control points such that the corner point $\mathbf{p}_{1,1}$ is located in the upper left region of the sketch. In total, we create 270k data pairs and 54k data pairs for the training and test sets, respectively. Among them, 80k training pairs and 16k test pairs contain occluded regions.

Network training. We implement the networks using PyTorch [53]. We first train VD-Net, OD-Net, and BSR-Net in a sequential way. The VD-Net is first trained for 30 epochs and fixed. Then the OD-Net is trained for 20 epochs and fixed. Afterwards, the BSR-Net is trained for 50 epochs. The CDF-Net is trained separately for 30 epochs. Each network is trained on a single Nvidia V100 GPU, using the Adam optimizer [54] with a start leaning rate of 10^{-4} and a batch size of 48. The training of VD-Net and OD-Net takes about 16 hours each, and the training of BSR-Net and CDF-Net takes about 10 hours and 8 hours respectively. Finally, VD-Net, OD-Net and BSR-Net are trained together for 20 epochs, with the learning rate for VD-Net and OD-Net reduced to 10^{-5}.
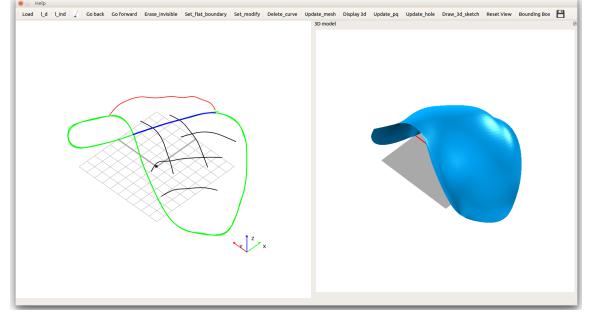
5 EXPERIMENTS AND USER STUDIES

We design a series of experiments and user studies to validate the effectiveness of our system. For intuitive interaction, the UI of our system includes a canvas for sketching and a 3D rendering pane for visualization of the result (see Fig. 10). In the canvas, the user can select the stroke type and draw the lines using a mouse or a stylus pen. The 3D rendering pane is updated in real-time to show the resulting B-spline surface, and the user can change the view direction to inspect the 3D result. Depth sample points can be specified on either the canvas view or the 3D view, and their depth values can be modified using the mouse scroll wheel. When the user is satisfied with the shape, they can use a meshing button to generate the PQ mesh. Examples of user interaction are shown in the accompanying video[1]. The system is tested on a PC with an Intel Core i5-8600K CPU at 3.6GHz, an Nvidia GeForce GTX 1060, and 16GB RAM. On average, the inference takes about 40ms for VD-Net, OD-Net and BSR-Net, and about 28ms for CDF-Net. The B-spline surface refinement takes about 100 ms. The PQ mesh extraction takes about 3.8s; since it only needs to be executed occasionally, this does not affect the interactivity of the system.

5.1 Evaluation

We evaluate the quality of the results generated by our system from two aspects. First, we measure the difference between the recovered B-spline surface and the ground-truth surface using their Chamfer distance based on point positions and normals, denoted as D_c and D_n respectively:

$$D_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{S}} \|\mathbf{x}_i - \mathbf{y}_{\alpha_i}\|^2 + \sum_{\mathbf{y}_j \in \mathcal{S}_{gt}} \|\mathbf{y}_j - \mathbf{x}_{\beta_j}\|^2}{|\mathcal{S}| + |\mathcal{S}_{gt}|}, \quad (13)$$

$$D_n = \frac{1}{|\mathcal{S}| + |\mathcal{S}_{gt}|} [\sum_{\mathbf{x}_i \in \mathcal{S}} d_n(N(\mathbf{x}_i), N(\mathbf{y}_{\alpha_i}))$$
$$+ \sum_{\mathbf{y}_j \in \mathcal{S}_{gt}} d_n(N(\mathbf{y}_j), N(\mathbf{x}_{\beta_j}))]. \quad (14)$$

Here $\mathcal{S}, \mathcal{S}_{gt}$ are the sample point sets for the recovered surface and the ground-truth surface respectively, $\mathbf{y}_{\alpha_i}$ is the closest point to $\mathbf{x}_i$ on the ground-truth surface, $\mathbf{x}_{\beta_j}$ is the closest point to $\mathbf{y}_j$ on the recovered surface, $N(\cdot)$ denotes the normal vector at a point, and d_n is the angle between two vectors:

$$d_n(\mathbf{v}_1, \mathbf{v}_2) = \arccos\left(|\mathbf{v}_1 \cdot \mathbf{v}_2| / (\|\mathbf{v}_1\| \cdot \|\mathbf{v}_2\|)\right). \quad (15)$$

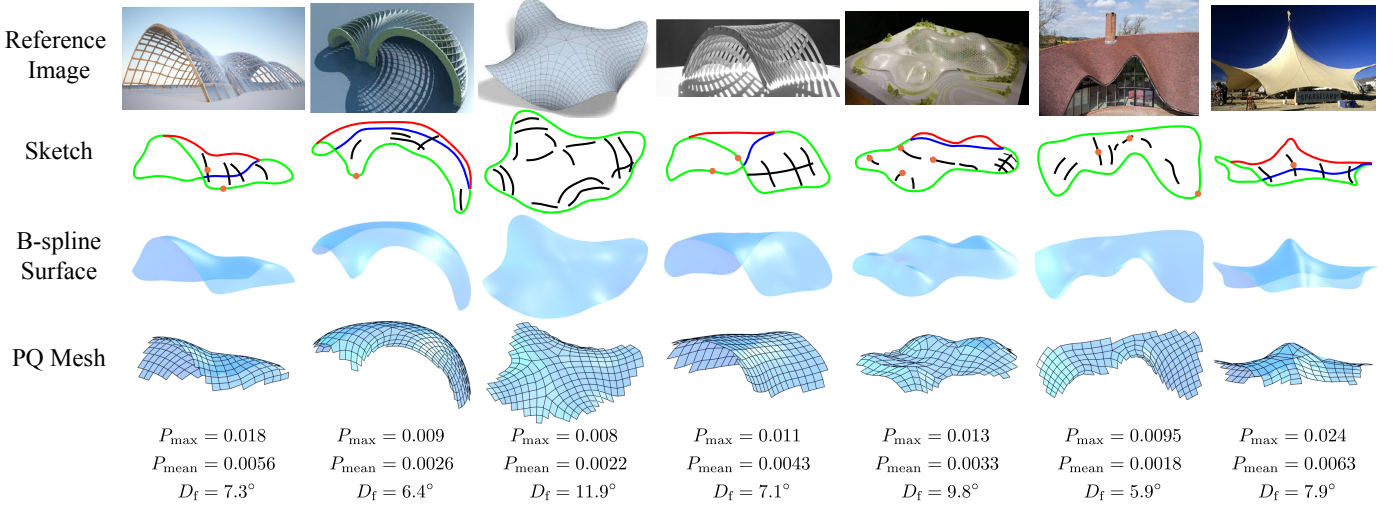1. https://www.youtube.com/watch?v=J-YYgaUd1hY

Fig. 11. Results on sketches drawn using freeform architecture images as reference. For each generated PQ mesh, the max and mean planarity errors $P_{\max}, P_{\mean}$, and the feature alignment error D_{f} (Eq. (16)) are also shown.

As there is no public sketch dataset for PQ meshes, the evaluation is performed using the test dataset created in Sec. 4. Table 1 shows the average values of D_{c} and D_{n} using our method on the test dataset. Since the sketches are represented as RGB images, for comparison we also include results from alternative learning-based methods [36], [55], [56] that can generate a disk-topology surface from a single sketch image. Among them, AtlasNet [55] represents the generated shape as a collection of parametric surface patches, and uses a loss function that penalizes the chamfer distance between sample point sets from the generated surface and the ground-truth surface. We test it using one, two and five patches, respectively. We also test a variant that uses a single patch and with additional loss terms to align the boundaries of the generated and ground-truth surfaces (see Appendix C). [56] uses a similar representation as AtlasNet, with additional loss terms to enforce consistency between the patches. We test it using one and four patches, respectively. [36] represents the shape with a template consisting of Coons patches. We test it using 10×10 bi-cubic Coons patches. All three methods are re-trained using our training dataset (see Appendix F). Table 1 shows their average values of D_{c} and D_{n} on our test dataset. Appendix C further shows two example sketches from the test dataset and the resulting surfaces using each method. These results show that our method is effective in recovering the underlying B-spline surface shape, and produces more accurate results than the alternative approaches.

In addition, we evaluate the quality of the resulting PQ mesh with two metrics. We measure the planarity error of each face using the distance between its two diagonal lines divided by the average edge length of the mesh, and calculate the maximum planarity error $P_{\max}$ and the mean planarity error $P_{\mean}$ across the whole mesh. We also measure the alignment between the PQ mesh edge directions and the feature lines from the input sketch. To do so, we first project the visible PQ mesh edges onto the viewing plane. Then we densely sample a set of points $\mathcal{S}_{\mathrm{f}} = \{\mathbf{f}_i\}$ from the feature lines, and compute an angle-based alignment error:

$$D_{\mathrm{f}} = \sum\nolimits_{\mathbf{f}_i \in \mathcal{S}_{\mathrm{f}}} d_{\mathrm{n}}(T(\mathbf{f}_i), H(\mathbf{g}_{\zeta_i})), \qquad (16)$$

TABLE 1
Average values of D_{c} and D_{n} (Eqs. (13) & (14)) on our test dataset using different methods to predict the underlying surface. For [55] and [56], the numbers in the sub-column headings indicate the number of patches.

	AtlasNet [55]				[56]		[36]	Ours
	1	2	5	variant	1	4		
D_{c} ($\times 10^{-2}$)	4.69	4.01	3.42	3.83	6.38	9.45	7.53	**0.98**
D_{n}	27.5°	22.5°	21.6°	18.7°	18.6°	21.5°	15.8°	**7.23°**

where $\mathbf{g}_{\zeta_i}$ is the closest point on the projected edges from the sample point $\mathbf{f}_i$, $T(\cdot)$ and $H(\cdot)$ denote the feature line tangent direction and the projected edge direction respectively, and d_{n} is the angle function defined in Eq. (15). On the test dataset, our method achieves the the average error metrics of $\overline{P}_{\max} = 0.0169$, $\overline{P}_{\mean} = 0.0042$ and $\overline{D}_{\mathrm{f}} = 7.8°$. It shows that our method can produce PQ meshes that align well with the input feature lines and with low planarity errors.

Figs. 11 and 12 show some examples of sketches outside our dataset, and their corresponding B-spline surfaces and PQ meshes generated with our system. The sketches in Fig. 11 are drawn according to rendered images of architectural shapes or real architectural photos, while the ones in Fig. 12 are drawn without reference shapes. In both figures, the resulting shapes align well with the structural lines and achieve the correct occlusion relation conveyed by the sketch. The figures also include the error metrics $P_{\max}, P_{\mean}$ and D_{f} for the PQ meshes, which show that the meshes have low planarity errors and align well with the feature lines.

In our system, the feature lines and depth samples enable the user to fine-tune the surface shape and the PQ mesh layout. Figs. 13 and 14 provide examples of such controls. Fig. 13 shows three inputs with the same sketch but different depth samples: the first one has no depth sample, while the other two have depth samples at the same locations but with different depth values. We can see that the depth values of the resulting PQ meshes are consistent with the specified values at the depth sample locations, which shows the effectiveness of depth samples for shape control. Fig. 16
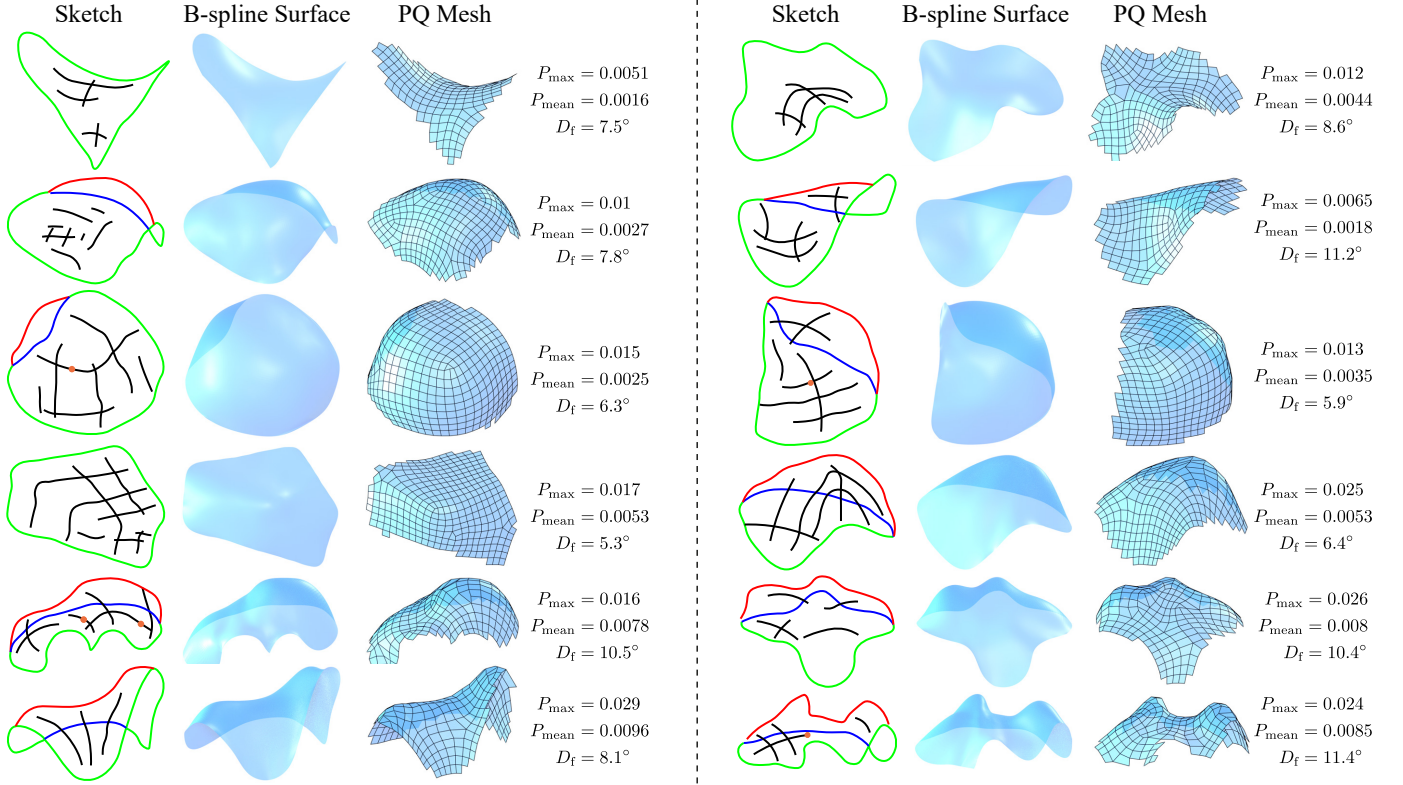
Sketch	B-spline Surface	PQ Mesh

$P_{max} = 0.0051$
$P_{mean} = 0.0016$
$D_f = 7.5°$

$P_{max} = 0.01$
$P_{mean} = 0.0027$
$D_f = 7.8°$

$P_{max} = 0.015$
$P_{mean} = 0.0025$
$D_f = 6.3°$

$P_{max} = 0.017$
$P_{mean} = 0.0053$
$D_f = 5.3°$

$P_{max} = 0.016$
$P_{mean} = 0.0078$
$D_f = 10.5°$

$P_{max} = 0.029$
$P_{mean} = 0.0096$
$D_f = 8.1°$

Sketch	B-spline Surface	PQ Mesh

$P_{max} = 0.012$
$P_{mean} = 0.0044$
$D_f = 8.6°$

$P_{max} = 0.0065$
$P_{mean} = 0.0018$
$D_f = 11.2°$

$P_{max} = 0.013$
$P_{mean} = 0.0035$
$D_f = 5.9°$

$P_{max} = 0.025$
$P_{mean} = 0.0053$
$D_f = 6.4°$

$P_{max} = 0.026$
$P_{mean} = 0.008$
$D_f = 10.4°$

$P_{max} = 0.024$
$P_{mean} = 0.0085$
$D_f = 11.4°$

Fig. 12. Results on sketches drawn without reference images. For each generated PQ mesh, the max and mean planarity errors P_{max}, P_{mean}, and the feature alignment error D_f (Eq. (16)) are also shown.
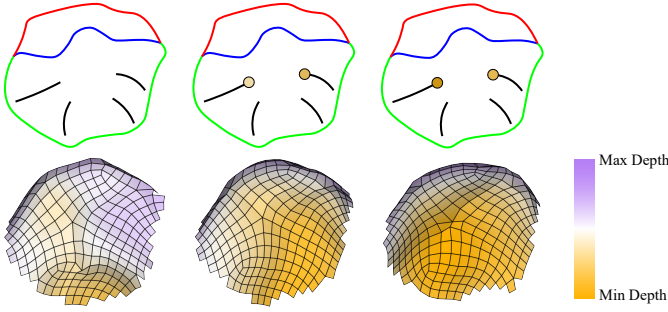

Fig. 13. Three sketches with the same strokes but different numbers of depth samples, and the resulting PQ meshes. Both the depth sample values and the PQ mesh depth values are visualized using color coding.

Fig. 14. An example where the resulting PQ mesh layout is fine-tuned by incrementally adding feature lines.

shows a series of sketches with the same structural lines and incrementally more feature lines. We can see that each additional feature line influences the PQ mesh shape and the mesh edge layout in its surrounding area, providing an intuitive tool to control the PQ mesh.

5.2 Ablation Study

Depth prediction. Our system predicts the depth and normal maps using two separate network modules for the visible regions and the occluded regions, respectively. We test an alternative network that predicts all the depth and normal maps jointly with a U-Net structure: the input sketch, depth samples and all the masks are fed to an encoder with the same architecture as the ones used in VD-Net and OD-Net; there are two decoder branches that generate
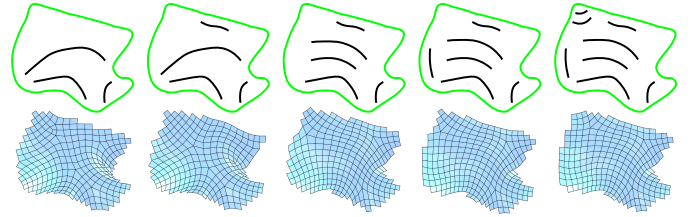
the visible and occluded depth maps separately, each with the same structure as the decoder in VD-Net and OD-Net, respectively. In addition, to validate the necessity of the orientation information in the masks, we also test an alternative approach that replaces the four input visibility-orientation masks to VD-Net and OD-Net with two masks that indicate the visible and occluded regions but not their orientations. We compare the depth maps generated by our method and the alternative approaches on the test dataset. For each method, we measure the accuracy of the predicted visible depth values $\{d_i^v \mid i \in \mathcal{P}_v\}$ and occluded depth values $\{d_j^o \mid j \in \mathcal{P}_o\}$ using their mean squared errors compared to the respective ground truth values $\{d_i^{v,gt}\}$ and $\{d_i^{o,gt}\}$:

$$D_v = \frac{\sum_{i \in \mathcal{P}_v}(d_i^v - d_i^{v,gt})^2}{|\mathcal{P}_v|}, \quad D_o = \frac{\sum_{j \in \mathcal{P}_o}(d_j^o - d_j^{o,gt})^2}{|\mathcal{P}_o|}. \quad (17)$$

Table 2 shows the average values of D_v and D_o on the test dataset for each method. It verifies that our method generates more accurate depth maps than the alternative approaches.

TABLE 2
Average values of depth error metrics D_v, D_o (Eq. (17)) on the test dataset using our method and two alternative methods: joint inference of visible and occluded depth maps ("Joint"), and using visible and occluded masks without orientation information ("Two-Masks").

	Ours	Joint	Two-Masks
D_v ($\times 10^{-2}$)	**0.6**	0.84	1.6
D_o ($\times 10^{-2}$)	**0.12**	0.64	1.2

TABLE 3
Average values of D_c, D_n (Eqs. (13) & (14)) on the test dataset using our B-spline surface prediction method and two alternative methods: direct prediction of control points using a ResNet, and using the BSR-Net alone without further optimization.

	Ours	ResNet	BSR-Net Only
D_c ($\times 10^{-2}$)	**0.98**	12.7	1.98
D_n ($\times 10^{-2}$)	**7.23°**	20.2°	12.3°

TABLE 4
Average values of CDF smoothness (E_{smooth}), max/mean PQ mesh planarity errors (P_{max}/P_{mean}), and feature line alignment error (D_f) on the test set, with and without the term $E_{cdf,cons}$ in the CDF-Net loss function.

	E_{smooth}	P_{max}	P_{mean}	D_f
With $E_{cdf,cons}$ (Ours)	**637**	**0.0169**	**0.0042**	**7.8°**
Without $E_{cdf,cons}$	775	0.0174	0.0043	8.9°

Our approach first handles the visible depth map, which is typically an easier problem than the occluded depth map due to its larger area as well as the feature lines that provide geometric information. Afterwards, the predicted visible depth map provides more cues for predicting the occluded depth map. We note that similar approaches have been adopted in existing 3D reconstruction methods such as [6], [57]. In addition, the orientation information is an important cue for the surface shape and its relation with the viewing plane. Hence the inclusion of orientation in the input masks for OD-Net and VD-Net improves the prediction accuracy.

B-spline surface prediction. Our system first uses BSR-Net to infer an initial B-spline surface and further optimizes it to align with the predicted depth maps. To validate the necessity of this setup, we test two alternative approaches: (1) using BSR-Net to infer the B-spline control points but without the subsequent optimization; (2) using a ResNet-18 [48] to directly infer the control points from the sketch without using the predicted depth maps. Table 3 shows the average values of the B-spline surface error metrics D_c, D_n (see Eqs. (13) and (14)) on the test dataset using each method. Both alternative approaches lead to less accurate results than our method. Fig. 15 shows some sketches from the test dataset and their resulting surfaces using each approach. We can see that direct inference with ResNet may produce surfaces that deviate from the structural lines. The BSR-Net, with the help of the predicted depth maps, can better align the surfaces with the structural lines than ResNet, although there can still be deviations in some local regions. The optimization after BSR-Net further reduces the local deviation and produces a result that aligns well with the structural lines.

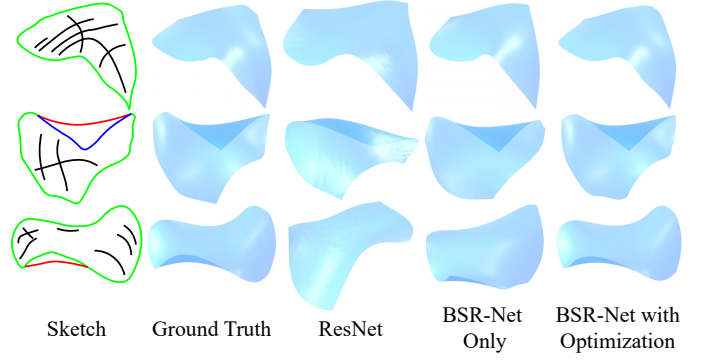Sketch · Ground Truth · ResNet · BSR-Net Only · BSR-Net with Optimization

Fig. 15. Comparison of the predicted B-spline surfaces using a ResNet, the BSR-Net alone, and the BSR-Net with further optimization. BSR-Net with optimization produces more accurate results.
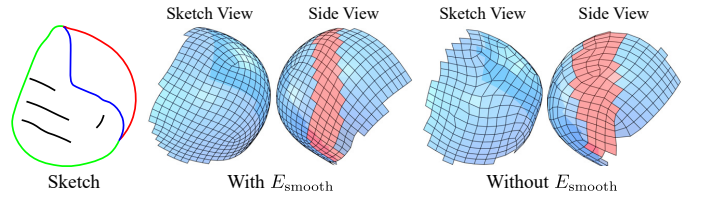

Sketch · With E_{smooth} · Without E_{smooth} (Sketch View / Side View)

Fig. 16. PQ mesh results with and without the term $E_{cdf,cons}$ in the CDF-Net loss function. In this example, the lack of $E_{cdf,cons}$ leads to a more irregular layout around the contour (shown in red in the side views).

CDF prediction. Our loss function for the CDF-Net includes a term $E_{cdf,cons}$ that enforces consistency of the CDF across the contour line. We test an alternative approach that does not include this term in the loss function. For each method, we compute a smoothness measure E_{smooth} of the CDF on the predicted surface using the Dirichlet energy of its PolyVector complex polynomial coefficients as explained in [58]. Table 4 shows the average value of E_{smooth} using each method on the test dataset. We can see that discarding the term $E_{cdf,cons}$ results in a CDF that is less smooth on average. Fig. 16 further shows an example of PQ meshes from each method. It can be seen that the removal of $E_{cdf,cons}$ leads to a less smooth PQ mesh layout especially around the contour region. Table 4 also shows the average values of the max and mean planarity errors P_{max}, P_{mean} and the alignment error D_f (see Eq. (16)) for the resulting PQ meshes using each method. It shows that the lack of smoothness for the CDF also affects the quality of the PQ meshes and increases the planarity errors and the deviation from the sketched feature lines.

5.3 User Studies

We conducted user studies to evaluate the usability of our system. The participants were 12 senior undergraduate or graduate architecture students with experience in 3D modeling software. They were given a short tutorial for the system and practiced with a few examples. The training took about 20 minutes for each participant. Afterwards, the participants were asked to model three PQ meshes. The first two need to be modeled according to a given rendered im-

Reference 1 · Reference 2

TABLE 5
Average scores for user study questions. The individual scores can be found in Appendix B.

Question	Q1	Q2	Q3	Q4	Q5
Average Score	3.84	3.92	4	4.07	4.23

age of an architectural surface and a real architectural photo, respectively (see inset). The third one was required to have self-occlusion and the participants could design any shape they liked. On average, each participant spent 10, 15 and 15 minutes modeling the three shapes, respectively. The modeling time was spent mostly on fine-tuning the shape until the user was satisfied with the result. Fig. 17 shows the designs from two participants for these tasks. We can see that the users were able to create designs similar to the reference shapes. More results are shown in Appendix D.

In addition, we designed the following questions collect to feedback from the participants:

1) Is the system intuitive compared to other 3D modeling tools you have used, such as Blender, Sketch-up and Rhino3D?
2) Does the generated shape meet your expectations?
3) Are the stroke lines effective in helping you create a model?
4) Does the feature of drawing occluded regions help your modeling?
5) Do the feature lines help you control the PQ mesh layout?

For each question, each participant was required to choose an answer from five options corresponding to a score of 1 to 5, where a higher score indicates better agreement. The detailed options and the participants' answers can be found in Appendix B. Table 5 shows the average score for each question. Overall, the system received positive feedback.

The participants were also asked to provide free-text comments and suggestions for improvement. Many participants commented that the capability to model the surface and PQ mesh simultaneously from a sketch is an attractive feature compared with other 3D modeling software. The main suggestions include: (1) development of plugins to integrate our system into existing modeling software; (2) additional constraints on the structural properties (i.e., designing PQ meshes that are also structurally sound); (3) capability to design multiple surfaces in a single sketch.

6 CONCLUSION AND DISCUSSION

In this work, we present a sketch-based modeling system for freeform roof-like structures represented as PQ meshes. Our system allows the user to draw structural lines for the surface boundary and contours with annotation for occlusion, as well as sparse feature lines that indicate the directions of PQ mesh edges. Utilizing a neural network trained with a synthetic dataset, the system can robustly generate a PQ mesh from a single sketch. The effectiveness and usability of the system are verified with extensive experiments and user studies. Our learning-based system provides notable benefits in efficiency and can generate the underlying surface and its CDF in real-time; although some existing non-learning-based
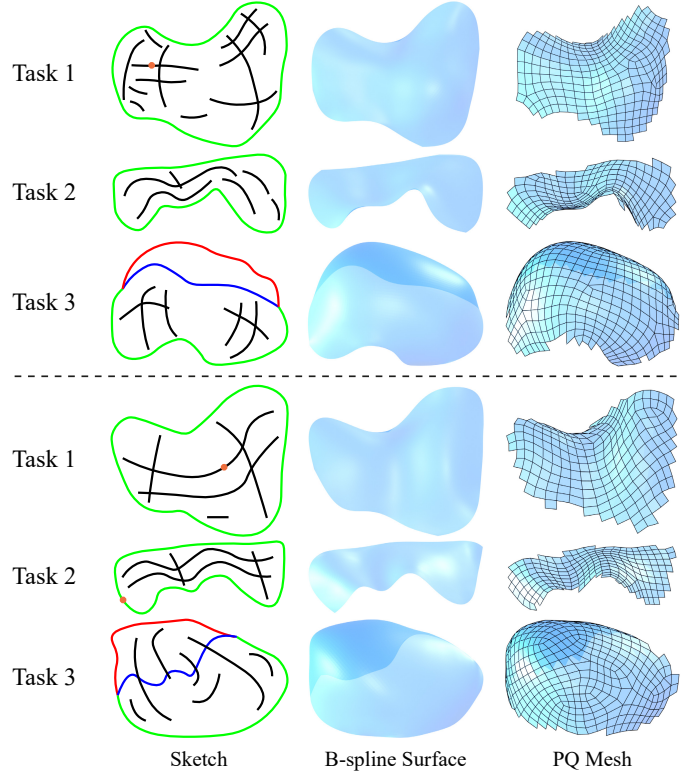

Fig. 17. The designs from two participants of the user study for the surface modeling tasks.

approaches may be adapted to achieve similar results, this would involve a large-scale non-linear optimization problem for *every* input sketch to find a surface and a CDF consistent with the sketch, which can be substantially slower. Our work shows that sketch-based modeling can be an effective tool for freeform architectural design where the components need to satisfy additional geometric constraints. It enables a novel design paradigm that complements the existing workflow, allowing the highly constrained 3D panel layout to be directly generated using its appearance from a certain view angle, without the need for an existing reference surface.

We envisage several avenues for future research that further improves the system. First, the system can be extended to handle more complex surface shapes and configurations such as non-disk topology and multiple layers of occlusions. These can potentially be achieved by adopting topology-agnostic surface representations such as signed distance fields [59] and adapting the network modules accordingly. Secondly, our current data generation procedure is based solely on the requirement of a smooth appearance. It can be further developed to incorporate other criteria such as structural properties and design preferences. Thirdly, due to the PQ meshing algorithm we use, the sketched boundary may not completely align with the boundary edges of the resulting PQ mesh. This can potentially be addressed by introducing optional constraints of boundary edge alignment in a future work. Fourthly, more fine-grained shape controls such as specification of singular vertices or sharp edges will be helpful additions to the system, which can potentially be achieved by augmenting the training data to include such elements. Finally, our approaches can be

extended to other types of structures and panel layouts in freeform architectural design, such as polyhedral surfaces and developable panels [1].

ACKNOWLEDGMENTS

REFERENCES

[1] H. Pottmann, M. Eigensatz, A. Vaxman, and J. Wallner, "Architectural geometry," *Computers & Graphics*, vol. 47, pp. 145–164, 2015.

[2] L. Olsen, F. F. Samavati, M. C. Sousa, and J. A. Jorge, "Sketch-based modeling: A survey," *Computers & Graphics*, vol. 33, no. 1, pp. 85–103, 2009.

[3] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, "FiberMesh: Designing freeform surfaces with 3D curves," *ACM Trans. Graph.*, vol. 26, no. 3, 2007.

[4] X. Chen, S. B. Kang, Y.-Q. Xu, J. Dorsey, and H.-Y. Shum, "Sketching reality: Realistic interpretation of architectural designs," *ACM Trans. Graph.*, vol. 27, no. 2, 2008.

[5] Y. Gingold, T. Igarashi, and D. Zorin, "Structured annotations for 2D-to-3D modeling," *ACM Trans. Graph.*, vol. 28, no. 5, 2009.

[6] J. Wu, C. Zhang, X. Zhang, Z. Zhang, W. T. Freeman, and J. B. Tenenbaum, "Learning 3D Shape Priors for Shape Completion and Reconstruction," in *ECCV*, 2018.

[7] Z. Lun, M. Gadelha, E. Kalogerakis, S. Maji, and R. Wang, "3D shape reconstruction from sketches via multi-view convolutional networks," in *3DV*, 2017, pp. 67–77.

[8] J. Delanoy, M. Aubry, P. Isola, A. A. Efros, and A. Bousseau, "3D sketching using multi-view deep volumetric prediction," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 1, no. 1, 2018.

[9] J. Glymph, D. Shelden, C. Ceccato, J. Mussel, and H. Schober, "A parametric strategy for freeform glass structures using quadrilateral planar facets," *Automation in Construction*, vol. 13, pp. 187–202, 2004.

[10] Y. Liu, H. Pottmann, J. Wallner, Y.-L. Yang, and W. Wang, "Geometric modeling with conical meshes and developable surfaces," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 681–689, 2006.

[11] Y. Liu, W. Xu, J. Wang, L. Zhu, B. Guo, F. Chen, and G. Wang, "General planar quadrilateral mesh design using conjugate direction field," *ACM Trans. Graph.*, vol. 30, no. 6, 2011.

[12] M. P. ao do Carmo, *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[14] F. Cordier, K. Singh, Y. Gingold, and M.-P. Cani, "Sketch-based modeling," in *SIGGRAPH ASIA 2016 Courses*, 2016.

[15] T. Igarashi, S. Matsuoka, and H. Tanaka, "Teddy: A sketching interface for 3d freeform design," in *Proceedings of SIGGRAPH*, 1999, p. 409–416.

[16] C.-L. Tai, H. Zhang, and J. C.-K. Fong, "Prototype modeling from sketched silhouettes based on convolution surfaces," *Comput. Graph. Forum*, vol. 23, no. 1, pp. 71–83, 2004.

[17] R. Schmidt, B. Wyvill, M. C. Sousa, and J. A. Jorge, "ShapeShop: Sketch-based solid modeling with blobtrees," in *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*, 2005.

[18] X. Chen, B. Neubert, Y.-Q. Xu, O. Deussen, and S. B. Kang, "Sketch-based tree modeling using markov random field," *ACM Trans. Graph.*, vol. 27, no. 5, 2008.

[19] A. Rivers, F. Durand, and T. Igarashi, "3D modeling with silhouettes," *ACM Trans. Graph.*, vol. 29, no. 4, 2010.

[20] E. Entem, L. Barthe, M.-P. Cani, F. Cordier, and M. Van De Panne, "Modeling 3D animals from a side-view sketch," *Computers & Graphics*, vol. 46, pp. 221–230, 2015.

[21] A. Jung, S. Hahmann, D. Rohmer, A. Begault, L. Boissieux, and M.-P. Cani, "Sketching folds: Developable surfaces from non-planar silhouettes," *ACM Trans. Graph.*, vol. 34, no. 5, 2015.

[22] M. Dvorožňák, D. Sýkora, C. Curtis, B. Curless, O. Sorkine-Hornung, and D. Salesin, "Monster mash: A single-view approach to casual 3d modeling and animation," *ACM Trans. Graph.*, vol. 39, no. 6, 2020.

[23] C. Li, H. Pan, Y. Liu, X. Tong, A. Sheffer, and W. Wang, "Bendsketch: Modeling freeform surfaces through 2d sketching," *ACM Trans. Graph.*, vol. 36, no. 4, 2017.

[24] H. Huang, E. Kalogerakis, E. Yumer, and R. Mech, "Shape synthesis from sketches via procedural models and convolutional networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 8, pp. 2003–2013, 2017.

[25] G. Nishida, I. Garcia-Dorado, D. G. Aliaga, B. Benes, and A. Bousseau, "Interactive sketching of urban procedural models," *ACM Trans. Graph.*, vol. 35, no. 4, 2016.

[26] X. Han, C. Gao, and Y. Yu, "Deepsketch2face: A deep learning based sketching system for 3D face and caricature modeling," *ACM Trans. Graph.*, vol. 36, no. 4, 2017.

[27] W. Su, D. Du, X. Yang, S. Zhou, and H. Fu, "Interactive sketch-based normal map generation with deep neural networks," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 1, no. 1, 2018.

[28] L. Wang, J. Wang, C. Qian, and Y. Fang, "Unsupervised learning of 3D model reconstruction from hand-drawn sketches," in *ACM Multimedia Conference*, 2018, pp. 1820–1828.

[29] C. Li, H. Pan, Y. Liu, A. Sheffer, and W. Wang, "Robust flow-guided neural prediction for sketch-based freeform surface modeling," *ACM Trans. Graph.*, vol. 37, no. 6, 2018.

[30] D. Du, X. Han, H. Fu, F. Wu, Y. Yu, S. Cui, and L. Liu, "SAniHead: Sketching animal-like 3D character heads using a view-surface collaborative mesh generative network," *IEEE Transactions on Visualization and Computer Graphics*, 2020.

[31] G. Yan, Z. Chen, J. Yang, and H. Wang, "Interactive liquid splash modeling by user sketches," *ACM Trans. Graph.*, vol. 39, no. 6, 2020.

[32] Z. Han, B. Ma, Y.-S. Liu, and M. Zwicker, "Reconstructing 3d shapes from multiple sketches using direct shape optimization," *IEEE Transactions on Image Processing*, vol. 29, pp. 8721–8734, 2020.

[33] C. Li, H. Pan, A. Bousseau, and N. J. Mitra, "Sketch2CAD: Sequential CAD modeling by sketching in context," *ACM Trans. Graph.*, vol. 39, no. 6, 2020.

[34] L. Yang, J. Wu, J. Huo, Y.-K. Lai, and Y. Gao, "Learning 3d face reconstruction from a single sketch," *Graphical Models*, vol. 115, 2021.

[35] S.-H. Zhang, Y.-C. Guo, and Q.-W. Gu, "Sketch2Model: View-aware 3d modeling from single free-hand sketches," in *CVPR*, 2021, pp. 6012–6021.

[36] D. Smirnov, M. Bessmeltsev, and J. Solomon, "Learning manifold patch-based representations of man-made shapes," in *International Conference on Learning Representations (ICLR)*, 2021.

[37] H. Pottmann, Y. Liu, J. Wallner, A. Bobenko, and W. Wang, "Geometry of multi-layer freeform structures for architecture," *ACM Trans. Graph.*, vol. 26, no. 3, 2007.

[38] M. Zadravec, A. Schiftner, and J. Wallner, "Designing quad-dominant meshes with planar faces," *Comput. Graph. Forum*, vol. 29, no. 5, pp. 1671–1679, 2010.

[39] O. Diamanti, A. Vaxman, D. Panozzo, and O. Sorkine-Hornung, "Designing N-PolyVector fields with complex polynomials," *Comput. Graph. Forum*, vol. 33, no. 5, pp. 1–11, 2014.

[40] Y.-L. Yang, Y.-J. Yang, H. Pottmann, and N. J. Mitra, "Shape space exploration of constrained meshes," *ACM Trans. Graph.*, vol. 30, no. 6, 2011.

[41] A. Vaxman, "Modeling polyhedral meshes with affine maps," *Comput. Graph. Forum*, vol. 31, no. 5, pp. 1647–1656, 2012.

[42] X. Zhao, C.-C. Tang, Y.-L. Yang, H. Pottmann, and N. J. Mitra, "Intuitive design exploration of constrained meshes," in *Advances in Architectural Geometry 2012*, L. Hesselgren, S. Sharma, J. Wallner, N. Baldassini, P. Bompas, and J. Raynaud, Eds. Vienna: Springer Vienna, 2013, pp. 305–318.

[43] B. Deng, S. Bouaziz, M. Deuss, J. Zhang, Y. Schwartzburg, and M. Pauly, "Exploring local modifications for constrained meshes," *Comput. Graph. Forum*, vol. 32, no. 2pt1, pp. 11–20, 2013.

[44] A. Vaxman, "A projective framework for polyhedral mesh modelling," *Comput. Graph. Forum*, vol. 33, no. 8, pp. 121–131, 2014.

[45] B. Deng, S. Bouaziz, M. Deuss, A. Kaspar, Y. Schwartzburg, and M. Pauly, "Interactive design exploration for constrained meshes," *Computer-Aided Design*, vol. 61, pp. 13 – 23, 2015.

[46] R. Poranne, R. Chen, and C. Gotsman, "On linear spaces of polyhedral meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 5, pp. 652–662, 2015.

[47] R. Adams and L. Bischof, "Seeded region growing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 641–647, 1994.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[49] G. E. Farin and D. Hansford, *The essentials of CAGD*. A K Peters, 2000.

[50] M. Deuss, A. H. Deleuran, S. Bouaziz, B. Deng, D. Piker, and M. Pauly, "Shapeop–a robust and extensible geometric modelling paradigm," in *Modelling Behaviour*. Springer, 2015, pp. 505–515.

[51] T. W. Sederberg and S. R. Parry, "Free-form deformation of solid geometric models," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, p. 151–160, 1986.

[52] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella, "Suggestive contours for conveying shape," *ACM Trans. Graph.*, vol. 22, no. 3, 2003.

[53] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[55] T. Groueix, M. Fisher, V. G. Kim, B. Russell, and M. Aubry, "A papier-mâché approach to learning 3D surface generation," in *CVPR*, 2018, pp. 216–224.

[56] Z. Deng, J. Bednařík, M. Salzmann, and P. Fua, "Better patch stitching for parametric surface reconstruction," in *8th International Conference on 3D Vision, 3DV 2020, Virtual Event, Japan, November 25-28, 2020*, 2020, pp. 593–602.

[57] J. Wu, Y. Wang, T. Xue, X. Sun, W. T. Freeman, and J. B. Tenenbaum, "MarrNet: 3D shape reconstruction via 2.5D sketches," in *Advances in Neural Information Processing Systems*, 2017.

[58] O. Diamanti, A. Vaxman, D. Panozzo, and O. Sorkine-Hornung, "Integrable PolyVector fields," *ACM Trans. Graph.*, vol. 34, no. 4, 2015.

[59] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," in *CVPR*, 2019, pp. 165–174.

Hao Pan is a senior researcher at Internet Graphics Group, Microsoft Research Asia. He received the BEng degree in software engineering from the Shandong University, and the PhD degree in computer science from The University of Hong Kong. His research interests include computer graphics, 3d computer vision, and geometry modeling and processing.

Wassim Jabi is a professor at the Welsh School of Architecture, Cardiff University where he leads the digital design area. He earned his B.Arch. from the American University of Beirut and his M.Arch. and Ph.D. from the University of Michigan. His current research is at the intersection of parametric design, the representation of space, building performance simulation, and robotic fabrication in architecture. He is a member of the editorial board of the International Journal of Architectural Computing (IJAC).

Juyong Zhang is a professor in the School of Mathematical Sciences at University of Science and Technology of China. He received the BS degree from the University of Science and Technology of China in 2006, and the PhD degree from Nanyang Technological University, Singapore. His research interests include computer graphics, computer vision, and numerical optimization. He is an associate editor of IEEE Transactions on Multimedia and The Visual Computer.

Bailin Deng is a senior lecturer in the School of Computer Science and Informatics at Cardiff University. He received the BEng degree in computer software (2005) and the MSc degree in computer science (2008) from Tsinghua University (China), and the PhD degree in technical mathematics (2011) from Vienna University of Technology (Austria). His research interests include geometry processing, numerical optimization, computational design, and digital fabrication. He is a member of the IEEE.

Zhi Deng is a PhD student at the School of Data Sciences, University of Science and Technology of China. His research interests include Computer Vision and Computer Graphics.

Yang Liu is a principal researcher at Microsoft Research Asia. He received his Ph.D. degree from The University of Hong Kong, Master and Bachelor degrees from University of Science and Technology of China. His recent research focuses on geometric computation and learning-based geometry processing and generation. He is an associate editor of IEEE Transactions on Visualization and Computer Graphics.

APPENDIX A
PROOF FOR PROPOSITION 1

We first prove the following lemmas.

Lemma 1. *If two minimal regions share a boundary line, then one of them must correspond to an occluded part of the surface.*

Proof. Assume that neither of the minimal regions corresponds to an occluded part. Then each of them corresponds to a separate visible region, and the shared boundary line corresponds to two separate boundary segments from the two regions. This will contract with Assumption 1(e). Therefore, at least one of the minimal regions must correspond to an occluded part of the surface. □

Lemma 2. *If a minimal region corresponds to an occluded part of the surface, then it must have an opposite orientation as the visible part that corresponds to the same minimal region.*

Proof. According to Assumption 1(d), the minimal region must be incident with a contour line. According to Assumption 1(c), the contour line corresponds to a curve on the surface that connects the visible part and the occluded part associated with the minimal region. Therefore, the visible part and the occluded part must have opposite orientations. □

We can now prove Proposition 1:

Proof. Suppose two minimal regions S_1 and S_2 share a boundary line. According to Lemma 1, one of them must correspond to an occluded part of the surface. Without loss of generality, we assume that S_1 corresponds to an occluded part S_1^o. We further denote the visible surface parts for S_1 and S_2 as S_1^v and S_2^v, respectively.

We first note that S_2^v must be connected with either S_1^o or S_1^v. Otherwise, due to Assumption 1(c), the shared boundary line must also correspond to a boundary segment of S_2^v, which will contract with Assumption 1(e).

If S_1 and S_2 share a visible boundary line, then S_2^v must be connected with S_1^o. Therefore, S_1^o and S_2^v have the same orientation. From Lemma 2, S_1^v and S_2^v must have opposite orientations, which proves Proposition 1(a).

Similarly, if S_1 and S_2 share an occluded boundary line, then S_1^v and S_2^v must be connected, and they have the same orientation. This proves Proposition 1(b). □

APPENDIX B
USER STUDY QUESTIONS AND ANSWERS

Below are the answers that the participants can choose for each question in the user study. The number before each answer is its score.

Q1: Is the system intuitive compared to other 3D modeling tools you have used, such as Blender, Sketch-up, and Rhinoceros?
- 1: Not at all.
- 2: Below average level.
- 3: Average level.
- 4: Slightly above average level.
- 5: Well above average level.

Q2: Does the generated shape meet your expectations?
- 1: Not at all.
- 2: The shape is overall close to the expected shape but lacks many details.
- 3: The shape is overall close to the expected shape but lacks some details.
- 4: The shape meets most of my expectations.
- 5: The shape meets all my expectations.

Q3: Are the stroke lines effective in helping you create a model?
- 1: Not at all.
- 2: They help to control local panel layouts but also affect other areas severely.
- 3: They help to control local panel layouts well but can sometimes affect other areas.
- 4: They are effective in helping me create a model when used in conjunction with depth samples.
- 5: They are very effective in helping me create a model.

Q4: Does the feature of drawing occluded regions help your modeling?
- 1: Not at all.
- 2: It can generate occluded regions but with unreasonable shapes.
- 3: It can generate occluded regions but the shape is not always reasonable.
- 4: It can generate occluded regions that basically meet my expectations.
- 5: It can generate occluded regions that always meet my expectations very well.

Q5: Do the feature lines help you control the PQ mesh layout?
- 1: Not at all
- 2: They help to control the layout in some minor areas.
- 3: They help to control the layout roughly.
- 4: They help to control the layout and basically meet my requirements.
- 5: They help to control the layout and are more effective than existing modeling software.

Table 6 shows answers from all the participants and the average score for each question.

TABLE 6
Answers from all participants for the user study questions.

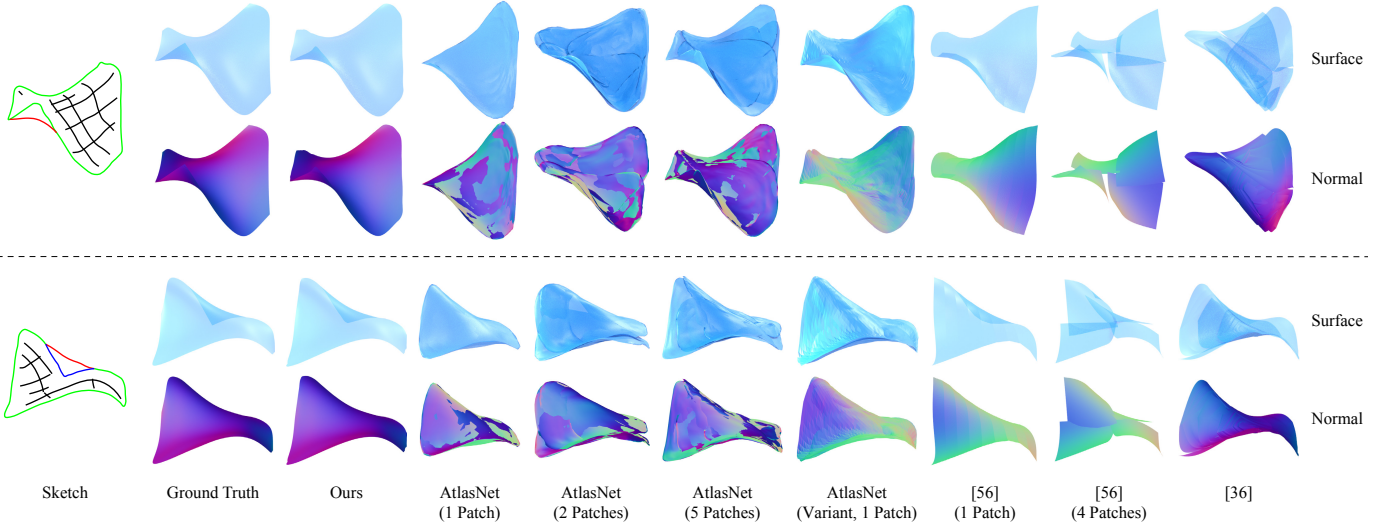| Score	Q1	Q2	Q3	Q4	Q5
I	4	5	4	5	4
II	3	3	4	5	4
III	5	5	5	5	5
IV	5	3	4	3	4
V	4	4	4	4	4
VI	3	4	3	3	4
VII	4	4	4	4	5
VIII	4	4	4	4	5
IX	4	4	4	5	4
X	4	4	5	4	4
XI	3	4	4	4	4
XII	3	3	3	4	4
Mean	3.84	3.92	4	4.07	4.23

Fig. 18. Two sketches from our test dataset, and the surface prediction results using our method, AtlasNet [55] (with one, two and five patches, and a variant with one patch), the method from [56] (with one and four patches), and the method from [36] (with 10×10 Coons patches).

APPENDIX C
COMPARISON OF SURFACE PREDICTION RESULTS

Fig. 18 shows two sketches from our test dataset, and the predicted surfaces using our method, AtlasNet [55], and the methods from [56] and [36]. We also implemented a variant of AtlasNet with a modified loss function: we sample the boundary and the interior of the generated surface and the ground-truth surface respectively, and replace the Chamfer distance term in the original loss function with a combination of two Chamfer distance terms computed using the boundary sample points and the interior sample points respectively; this variant provides better coverage of sample points on the surface boundary, which can improve the accuracy of the boundary shape. For each result, we also visualize the normals by rendering them as RGB colors over the surface. We can see that our method recovers a surface shape that is close to the ground truth, while the results produced from the other approaches deviate notably from the ground truth. Some results from the other approaches also exhibit incorrect fold-overs. This is potentially because their loss function only uses the Chamfer distance to align the generated shape with the ground truth: a fold-over shape with each "layer" close to the ground-truth shape may still achieve a low Chamfer distance.

APPENDIX D
MORE RESULTS FROM SURFACE MODELING TASKS

Fig. 19 shows the participants' designs for the three surface modeling tasks in the user studies, in addition to the ones shown in Fig. 17 in Section 5.3.

APPENDIX E
MORE SHAPES FROM OUR DATASET

Fig. 20 shows more examples of B-spline surfaces from our dataset.

APPENDIX F
NETWORK TRAINING DETAILS FOR COMPARISON WITH [36], [55], [56]

The networks of [55], [56] and [36] can learn the generation of surface shapes from images. We adopt this setting for all three methods, using their open-source implementation[2,3,4] for training and testing. All parameters are set to their default values given in the open-source implementations. For all three methods, the training requires sample point positions on the ground-truth surface. Some methods also require sample point normals. For this purpose, we regularly sample 100×100 parameter values over the parameter domain of the ground-truth B-spline surface, and use their corresponding point positions and normals on the surface for the training. Both [55] and [56] represented the generated shape as a collection of patches. We train three networks for [55] using one, two, and five patches, respectively. We also train two networks for [56] using one and four patches, respectively. In [36], the generated shape is represented using a template consisting of multiple Coons patches, and the default template has a sphere topology. Since we want to generate B-spline surfaces of disk topology, we follow the description in [36] to construct a disk-topology template consisting of 10×10 Coons patches and use it for the training.

2. https://github.com/ThibaultGROUEIX/AtlasNet
3. https://github.com/GentleDell/Better-Patch-Stitching
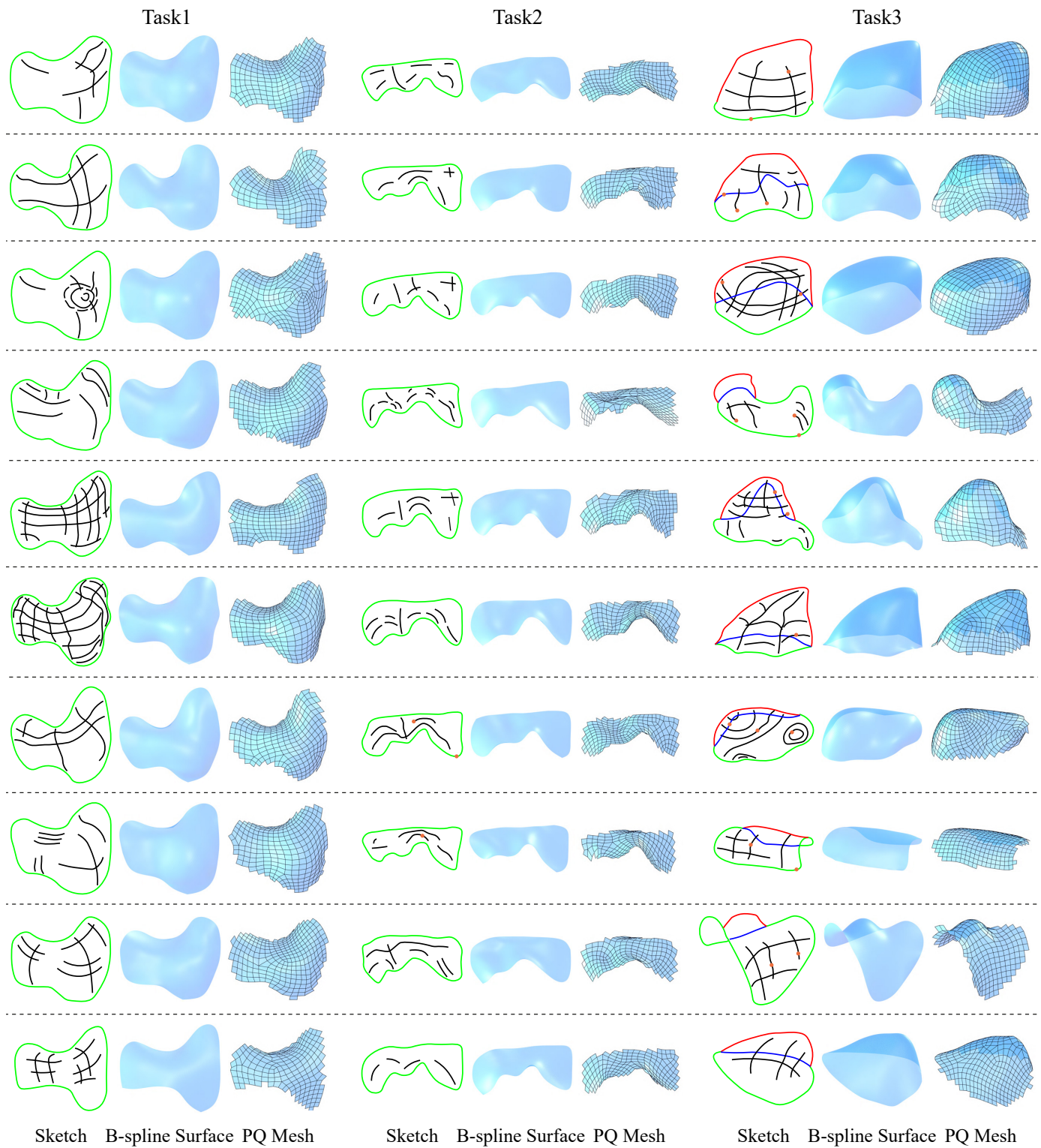4. https://github.com/dmsm/LearningPatches

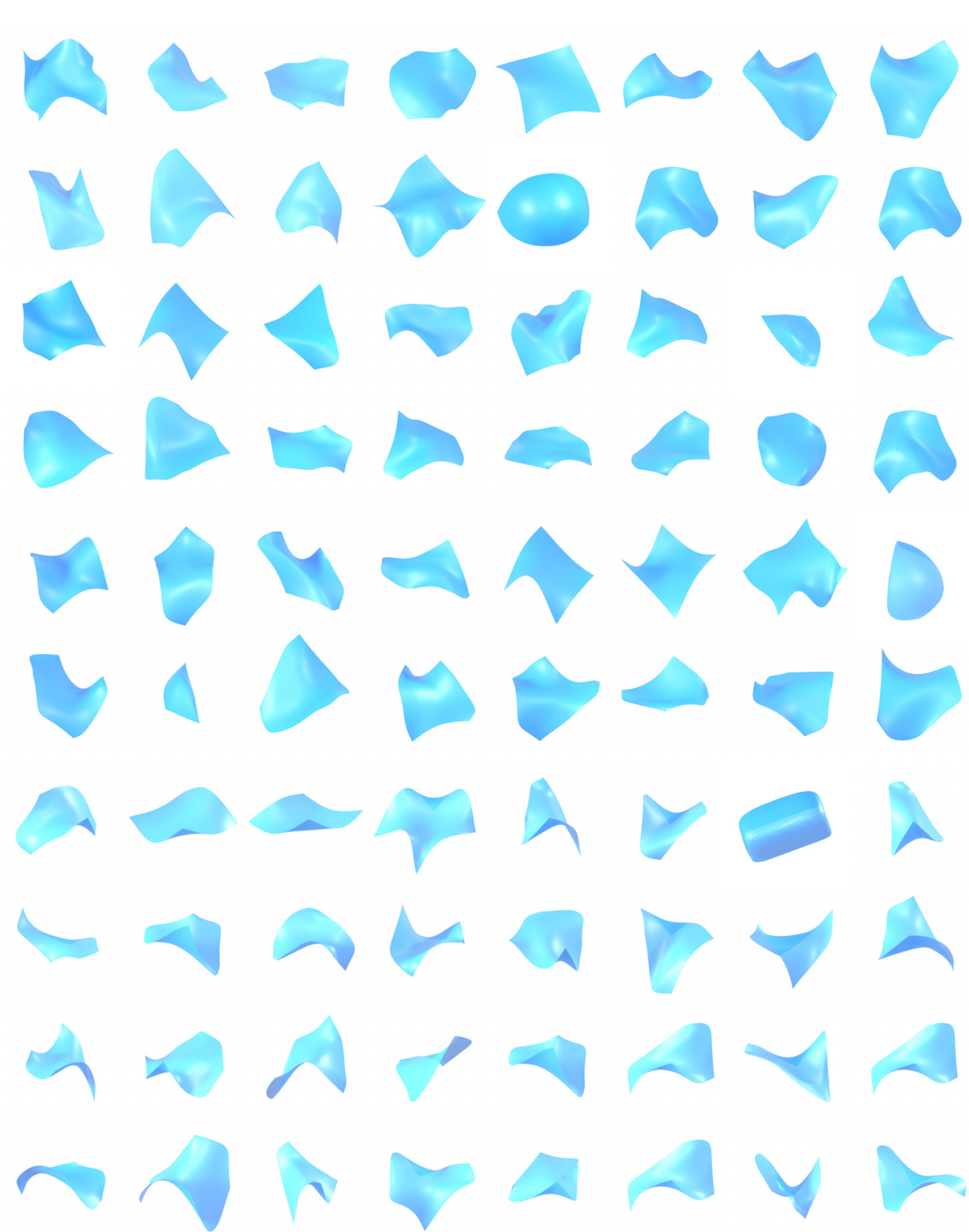Fig. 19. Results of the surface modeling tasks besides the ones already shown in Fig. 17. Each row shows the results from one participant

Fig. 20. More examples of B-spline surfaces from our dataset.