

# BendSketch: Modeling Freeform Surfaces Through 2D Sketching

CHANGJIAN LI, The University of Hong Kong and Microsoft Research Asia

HAO PAN, YANG LIU, and XIN TONG, Microsoft Research Asia

ALLA SHEFFER, University of British Columbia

WENPING WANG, The University of Hong Kong

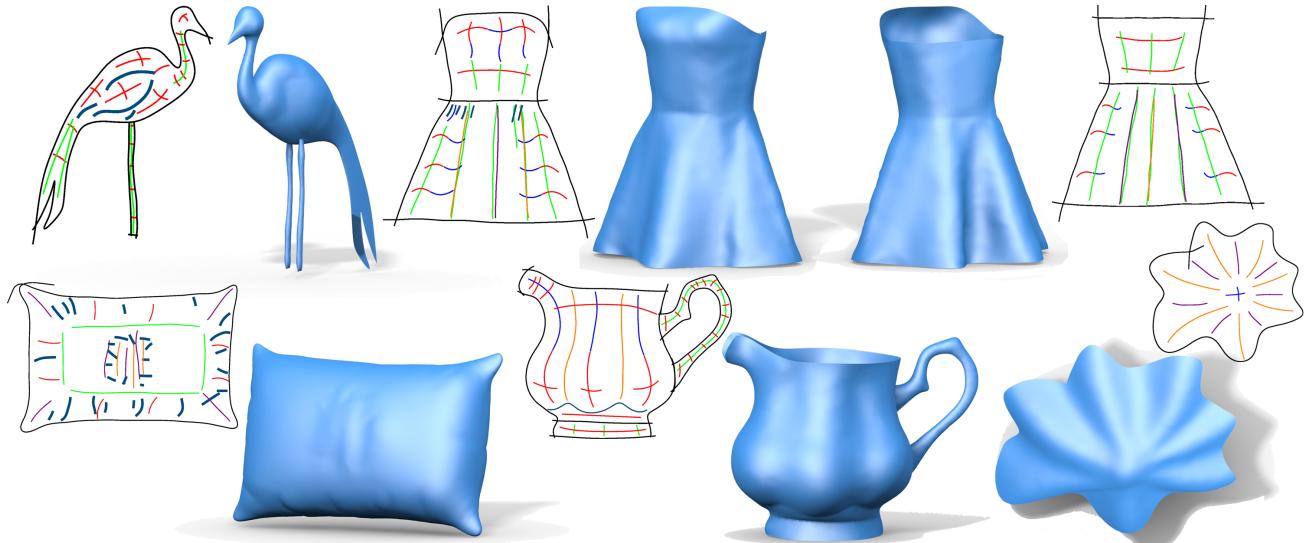


Fig. 1. Five freeform shapes created by our sketch-based modeling approach, with the corresponding planar sketches. The curvy container is a single surface patch, the crane, pillow and teapot are each made from two symmetric surface patches, and the skirt is made from two surface patches sketched individually. The annotations for the sketched strokes are defined in Sec. 3.2.

Sketch-based modeling provides a powerful paradigm for geometric modeling. Recent research had shown, sketch based modeling methods are most effective when targeting a specific family of surfaces. A large and growing arsenal of sketching tools is available for different types of geometries and different target user populations. Our work augments this arsenal with a new and powerful tool for modeling complex freeform shapes by sketching sparse 2D strokes; our method complements existing approaches in enabling the generation of surfaces with complex curvature patterns that are challenging to produce with existing methods.

To model a desired surface patch with our technique, the user sketches the patch boundary as well as a small number of strokes representing the major bending directions of the shape. Our method uses this input to generate a curvature field that conforms to the user strokes and then uses this field to

derive a freeform surface with the desired curvature pattern. To infer the surface from the strokes we first disambiguate the convex versus concave bending directions indicated by the strokes and estimate the surface bending magnitude along the strokes. We subsequently construct a curvature field based on these estimates, using a non-orthogonal 4-direction field coupled with a scalar magnitude field, and finally construct a surface whose curvature pattern reflects this field through an iterative sequence of simple linear optimizations.

Our framework is well suited for single-view modeling, but also supports multi-view interaction, necessary to model complex shapes portions of which can be occluded in many views. It effectively combines multi-view inputs to obtain a coherent 3D shape. It runs at interactive speed allowing for immediate user feedback. We demonstrate the effectiveness of the proposed method through a large collection of complex examples created by both artists and amateurs. Our framework provides a useful complement to the existing sketch-based modeling methods.

CCS Concepts: • Computing methodologies → Mesh models;

Additional Key Words and Phrases: sketch, freeform shape, curvature field, multi-view

## ACM Reference format:

Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. 2017. BendSketch: Modeling Freeform Surfaces Through 2D Sketching. *ACM Trans. Graph.* 36, 4, Article 125 (July 2017), 14 pages.

DOI: <http://dx.doi.org/10.1145/3072959.3073632>

This work is supported by National Basic Research Program of China (2011CB302400), NSFC (61272019), the Research Grant Council of Hong Kong (717012, 17209815, 17211017), sponsorship from Microsoft (AR160005) and NSERC.

Author's addresses: C. Li, Computer Science Department, The University of Hong Kong; H. Pan and Y. Liu and X. Tong, Microsoft Research Asia; A. Sheffer, Computer Science Department, University of British Columbia; W. Wang, Computer Science Department, The University of Hong Kong.

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

© 2017 ACM. 0730-0301/2017/7-ART125 \$15.00

DOI: <http://dx.doi.org/10.1145/3072959.3073632>

## 1 INTRODUCTION

Our natural environments contain many irregular, complex free form shapes: the clothing and shoes we wear, our bags, hats, soft furniture, dishes, animals and plants. While traditionally such geometries were modeled using CAD softwares, in recent years there is an increasing effort to use more user-friendly sketch-based modeling tools for this task. Unfortunately while existing sketching tools are suitable for modeling many classes of shapes (Section 2), they are not well suited for designing free form surface with complex curvature patterns that show up on natural shapes such as animal bodies, clothing, or even tableware (Figure 1). We present a method that allows users to model such varying surface curvature patterns using a sketching interface (Figure 1).

The local curvature variation of a freeform shape can be vividly described by a small set of curves showing the major bending directions on the surface. Our work takes such bending stroke patterns and uses them to derive detailed surface geometry. Our method takes as input a set of *sparse and unorganized bending strokes*, as well as other supplemental curves, and automatically infers the freeform shapes that follow these strokes. Bending strokes provide a natural interface, frequently used by artists for depicting details on curved surfaces (c.f. the works of Thomas Nast). In mathematical terms, these strokes approximate the 2D images of smooth principal curvature lines of the depicted 3D shapes. In computer graphics, such curves have been used as the hatching lines for non-photo-realistic rendering (NPR) of 3D shapes [Hertzmann and Zorin 2000], and as constraints for generating normal vector fields for 2.5D rendering of planar drawings [Iarussi et al. 2015; Shao et al. 2012]. Inspired by these methods we develop an algorithm that combines sketched bending curves with other characteristic surface curves such as ridge and valley lines, contours, and sharp feature curves, and uses these as input for modeling complex free-form surfaces.

We represent freeform surface patches as height-field surfaces; modeling such a patch in our system is equivalent to shape reconstruction from projected principal curvature lines approximated by the input bending strokes. The challenge of such reconstruction is two-fold, first we need to interpret the input resolving ambiguities and second we need a systemic approach for converting curvature information into surface geometry, an inherently non-linear problem. First, we a priori do not know if the sketched curves convey convex or concave bending directions of the 3D surface; this is an example of the well-known bas-relief ambiguity with 2D depictions of 3D shapes [Belhumeur et al. 1999]. Second, the amount of bending along the strokes is similarly unknown: the 2D shape of a stroke is the combined result of both its spatial shape on the surface and the relative orientation of this spatial curve with respect to the view or projection direction. Since the shape of the surface is unknown there is no unique way to infer the curve’s spatial shape (or how much the surface bends along the curve) solely from its 2D projection. Third, while the strokes provide some information on the local geometry in their immediate surroundings, they are surrounded by large blank regions that have no such information; our challenge is to propagate the stroke data across these regions in a smooth and meaningful way. The question amounts to modeling a curvature

tensor field of the spatial surface as it is projected onto the 2D plane, given the constraints of a sparse set of projected curvature lines.

To resolve the above mentioned challenges, we make the following technical contributions in this paper. To disambiguate the bending strokes representing convex and concave surface curvatures, we find convexity correlations among locally proximate strokes, and utilize the correlations to find the most consistent convexity labeling of the strokes. To estimate the real curvature of the unknown surface along the bending strokes, we jointly optimize both the curvature values along the strokes and the unknown surface geometry, converging to a common spatial shape. Lastly, to propagate geometric information from the sparse strokes across the rest of the surface we construct a smooth curvature field model, which is represented by an non-orthogonal 4-directional field coupled with a scalar magnitude field, from which the surface with desired curvature pattern is constructed through an iterative sequence of simple linear optimizations.

We extend our approach to the multi-view setting, to allow for modeling of more complex shapes: for each view, the method takes the sparse 2D user strokes as input, and builds the height field surface for the current view; as the user rotates to new views and sketches additional curves, the method builds partial surfaces for every view and combines them smoothly through shape preserving registration.

The effectiveness of the proposed method is demonstrated through a range of examples, an extensive user evaluation done with our system by novice users without artistic training who were able to successfully create a range of non-trivial geometries, shown in Section 5, and a detailed comparison to previous methods. We also validate the steps of our method through comparison with ground truth data.

## 2 RELATED WORK

Sketch-based modeling is an active research area spanning a large range of methods addressing different categories of shapes. For detailed surveys of both earlier works and more recent progresses, we refer the reader to [Company et al. 2005; Ding and Liu 2016; Jorge and Samavati 2011; Olsen et al. 2009]. Below we briefly discuss the major categories of sketch-based modeling methods based on the type of geometries they target and the type of curves they utilize.

*Contour based models.* Starting from the seminal Teddy system [Igarashi et al. 1999], a range of methods use contour curves to infer 3D geometric shape. They form smooth low-frequency 3D surfaces constrained by the contours, where the surfaces are constructed as membrane functionals [Igarashi et al. 1999; Joshi and Carr 2008; Nealen et al. 2007; Zhang et al. 2001], or defined by distance transforms and implicit functions [Bernhardt et al. 2008; Olsen et al. 2011; Schmidt et al. 2005; Tai et al. 2004]. The methods support adding more details to the models through marking positional, normal, or curvature value constraints, or by refining and blending the implicit functions, to produce sharp features and ridges/valleys, frequently leveraging multi-view and multi-layer frameworks. Karpenko et al. [2006] show how to infer the topology of a plausible solid shape given its visible contours drawn by users, and build the smooth shape through inflation which they model as the relaxation of a

mass-spring system. Yeh *et al.* [2016] model 3D objects with self-occlusions out of photos, by classifying the occlusion types and connecting separated patches, and lifting the shapes through marking the depth variance along patch boundaries and mean curvature values in the patch interiors.

Multiple frameworks specifically address modeling of smooth shapes whose parts exhibit high degrees of rotational symmetry. Exploiting the symmetry and simplicity of these geometric primitives, the user only needs to sketch a few profile curves to determine their specific shapes. For example, Gingold *et al.* [2009] have users annotate over the 2D sketch which parts are generalized ellipsoids, which parts are generalized cylinders, and how these parts relate to each other through symmetry, adjacency, etc. The various parts are then constructed and fitted together to form the complete 3D shape. Similarly in [Shtof *et al.* 2013], the user first draws a 2D outline of the object, and then incrementally selects geometric primitives from a set of candidates and places each object over the 2D sketch, while the method automatically fits the geometric primitives to the 2D sketches and also to previously placed parts for consistency. Chen *et al.* [2013] utilize generalized cylinders pervasively for the easy modeling of 3D objects from photographs, where the photos are first processed to extract object contours and then the user sketches the base and profile curves to guide the construction of generalized cylinders that also fit to the extracted contours. Andre *et al.* [2007] use two sets of strokes that depict orthogonal curves of the 3D surface, and recover the surface by sweeping one set of strokes along the other. A later work, [Andre and Saito 2011], has users sketch in a single view the two profile curves of a generalized ellipsoid and the contour curve; the system then constructs a 3D shape by sweeping one profile curve along the other while fitting to the contour. Miao *et al.* [2015] show the use of symmetry in constructing more generalized ellipsoids and in combining components into a full cartoon character.

Our framework complements these approaches by extending the sketching paradigm to smooth high-frequency surfaces by allowing users to describe curvature variation across surfaces and enriching the set of output models with additional smooth details. Indeed, while previous contour-based methods aim at creating smooth and plausible shapes with local sharp features, in our approach the direct specification of curvature variation patterns through sketching bending lines enables the creation of additional controlled complexity (see Figs. 1, 6), which can be outside of the scope of many previous methods or very tedious for them otherwise (Sec. 5.3).

**Curve networks.** A range of methods focus on extracting 3D curve geometry from artist strokes, assumed to represent meaningful surface curves [Bae *et al.* 2008; Schmidt *et al.* 2009; Xu *et al.* 2014]. They then create 3D shapes by interpolating the resulting curve networks using perception-driven surfacing methods [Bessmeltsev *et al.* 2012; Pan *et al.* 2015; Zhuang *et al.* 2013]. ILoveSketch [Bae *et al.* 2008] presents an interactive system that allows the sketching of 3D curves through a 2D interface. Schmidt *et al.* [2009] support sketching 3D curves through the incremental construction of scaffold structures. True2Form [Xu *et al.* 2014] converts a 2D curve network to 3D, by enforcing the various regularities inherent in many man-made objects, including the orthogonality of crossing

curves, the planarity of curves, the parallelism/orthogonality of object faces, etc. [Li *et al.* 2007] presents a closed form solution for reconstruction of piecewise planar objects from the 2D line drawing of edges of the objects; [Wang *et al.* 2009] extends the range by reconstructing shapes with curved faces from 2D line drawings, as it first recovers the 3D wireframe through regularities and then fills up the faces with Bezier patches or triangle meshes. These methods target man-made shapes which exhibit high-regularities and relatively low curvature variation. Our framework addresses the modeling of more natural and less regular shapes which exhibit complex curvature patterns.

**Domain specific methods.** Many sketching methods are domain-specific, e.g. targeting the creation of 3D characters [Bessmeltsev *et al.* 2015; Cordier *et al.* 2011; Entem *et al.* 2014], garments [Robson *et al.* 2011; Turquin *et al.* 2004], developable surfaces with folds [Jung *et al.* 2015; Zhu *et al.* 2013], layered models [De Paoli and Singh 2015] or hair [Fu *et al.* 2007]. These methods leverage domain specific cues and are not applicable to the more general setting we aim to address.

**Data-driven and learning-based methods.** Early data-driven methods for sketch-based modeling use the user sketch as a key for search through a database of shapes, and retrieve shapes whose features resemble the sketch; The retrieved models are subsequently deformed to better fit the contours specified in the sketches. Examples include scene modeling [Xu *et al.* 2013] and object modeling [Xie *et al.* 2013]. The obvious limitation of these methods is that the database has to be large and diverse to allow true flexibility in modeling.

Recently researchers also employ advanced machine learning techniques like Convolutional Neural Networks (CNN) to build a direct mapping from user sketches to 3D shapes, and specifically to parameters of procedural 3D shape models [Huang *et al.* 2016; Nishida *et al.* 2016]. While capable of providing impressive convenience to users through the powerful mapping modeled by CNNs, these methods are usually restricted to modeling particular classes of objects being observed through fixed viewpoints, as a neural network built for one object class under one viewpoint does not generalize to other situations.

**Drawing on 3D canvas.** A number of methods support the creation of simple 3D objects first, and then allow the user to incrementally edit the 3D object to add new components and details. For example, Teddy [Igarashi *et al.* 1999] and FiberMesh [Nealen *et al.* 2007] allow both the creation and editing of simple smooth shapes through sketching, while [Nealen *et al.* 2005] and [Gingold and Zorin 2008] let the user edit existing shapes through sketching new contours, suggestive contours and other shading-based feature lines. Our framework increases the stroke vocabulary supported by these systems allowing users to specify curvature variation across the surface directly, which can be tedious to produce with the positional and shading-based strokes.

3D modeling softwares, e.g. Zbrush[ZBr 2016], support sculpting and “drawing on 3D canvas” interaction, but require significant expertise to use these features effectively. Compared with these 3D drawing methods, the technique in this paper focuses on the

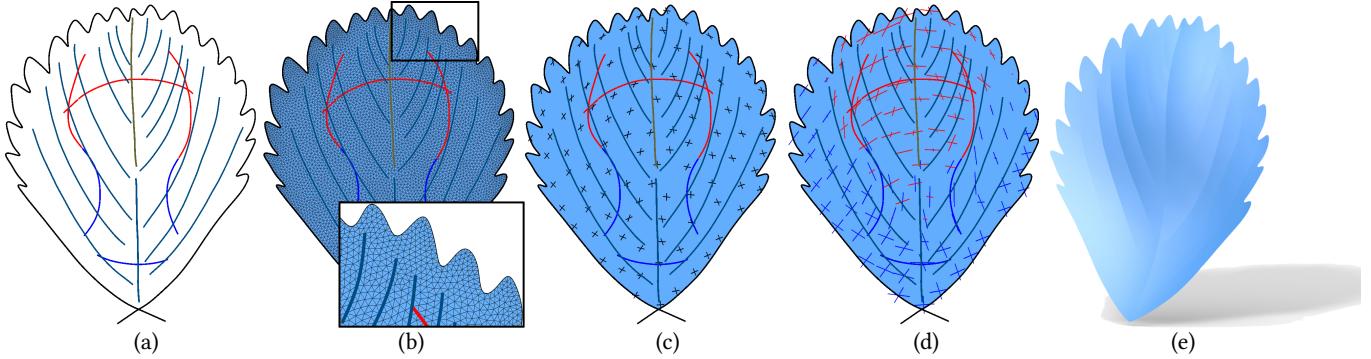


Fig. 2. Surface sketching pipeline for one view. From left to right: (a) the input strokes parsed with labeling of convexity (red/blue for convex/concave), (b) the 2D domain mesh, (c) the 2D 4-direction field modeling curvature directions, (d) the finally solved 2D curvature field with curvature directions and magnitudes (red/blue for positive/negative curvatures), and (e) the solved surface.

quick modeling of freeform shapes through simple 2D sketching consisting of very few strokes.

*Normals from sketches.* A range of methods address the reconstruction of normal vectors from sketches, using the normals to apply a range of rendering effects to the underlying sketches. In [Sýkora et al. 2014], a bas-relief 3D proxy is created to enable the rendering of 2D sketches with vivid 3D impressions; this proxy is not designed to provide a correct 3D reconstruction. CrossShade [Shao et al. 2012] leverage orthogonality between sketched curvature lines to infer normal vectors; many of their ideas were instrumental in lifting curve networks to 3D in [Xu et al. 2014], however the normals they produce are not accurate enough to facilitate actual reconstruction [Xu et al. 2014]. Similarly, Bui et al. [2015] compute smoothly varying normal vectors from silhouettes and hatch lines which are also regarded as curvature lines. Iarussi et al. [2015] introduce the notion of BendField energy to measure the smoothness of the projected curvature directions on 2D plane, and solve normal vectors as the cross product of lifted curvature directions; we are inspired by this approach and extend the BendField energy to model a complete curvature field projected on 2D, as presented in Sec. 3.4. Xu et al. [2015] infer normal vectors from a different type of curves, the so called isophotes, which represent curves of equal shading. The isophote curves are less intuitive than bending strokes for many people and they involve the specification of the lighting direction; on the other hand, isophote curves as constraints on surface curvature can be naturally incorporated into our framework (Sec. 3.4).

### 3 SINGLE VIEW MODELING

In this section, we focus on how to model a single freeform surface patch through sketching 2D bending strokes and other curves. In Section 4, we show the extension to multiple view processes for modeling more complex shapes with occlusion that cannot be described in a single view.

#### 3.1 Overview

Our framework takes as input a specific set of sketching strokes, and solves the surface patch that matches the input. The bending strokes and other strokes used in our framework are defined in Section 3.2. The computational process from sketch to surface consists of several steps (see Fig. 2) to be presented in the following sections:

- (a) the processing of input strokes, in particular to disambiguate the bending strokes from representing convex/concave surface curvature (Sec. 3.2),
- (b) the triangulation  $M$  of the sketched domain  $\Omega$  for subsequent surface computation (Sec. 3.3),
- (c) the principles of constructing a curvature field on the triangulated domain from bending strokes, and of reconstructing surface from the curvature field (Sec. 3.4), and
- (d) the algorithm of joint computation of the curvature field as well as the surface through an iterative process where both the curvature magnitudes and the surface shape are solved (Sec. 3.5).

Finally, in Sec. 3.6 we discuss the boundary conditions and how the other strokes affect surface shape computationally.

#### 3.2 2D Strokes

*3.2.1 Stroke types and representation.* In our framework, we use the following types of 2D strokes to model the freeform surfaces (Fig. 2(a)):

- *boundary* - where the surface discontinues,
- *bending stroke* - projection of a principal curvature line of the spatial surface, can represent **convex/concave** surface curvature,
- *sharp feature* - a curve across which the normal vectors change discontinuously with **convex/concave** angles,
- *ridge/valley* - a curve across which the surface always bends back/forward,
- *flat stroke* - a curve along which the surface has no bending,
- *contour* - boundary with known surface normal directions, or object silhouette.

The first three types of strokes are basic and orthogonal in function, while the ridge/valley strokes are extensions meant to ease the modeling of particular shapes and are actually implemented by the basic bending strokes. Flat strokes, a special case of bending strokes with vanishing curvature, are used for modeling almost developable surface regions that have no bending in the stroke direction.

All strokes are represented as 2D polylines. In our interactive sketching system, they are first acquired by tracing the pen motion as the user draws the 2D sketching, and then resampled uniformly and smoothed to remove noise.

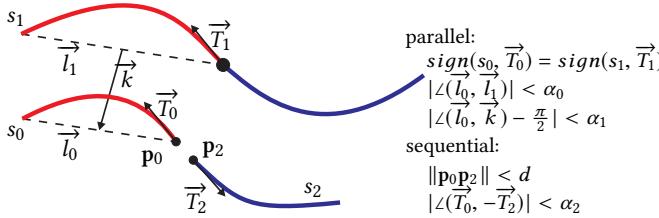


Fig. 3. Observations about the relations of convex/concave properties between neighboring curve segments. The lower two strokes  $s_0, s_2$  are sequential to each other, while the left two strokes  $s_0, s_1$  are parallel. The geometric quantities and thresholds used for checking the stroke relations are illustrated. In our experiments, we use  $\alpha_0 = \alpha_2 = 15^\circ$ ,  $\alpha_1 = 20^\circ$ ,  $d = D/30$ , and two simple strokes are considered neighbors if their closest distance is less than  $D/10$ , with  $D$  the diagonal length of the bounding box of sketch domain.

**3.2.2 Stroke parsing.** Given a set of 2D bending strokes scattered inside the sketching domain, we need to know whether the strokes represent convex surface bending or not, which is essentially about resolving the bas-relief ambiguity inherent to the planar depictions of a 3D shape. Rather than having the user manually label whether each stroke represents convex curvature or otherwise, we try to automatically decide it. For the rest types of strokes, the convexity of sharp features and the choice of ridge versus valley curves are specified by the user, as these curves represent marked features of which the user wants direct control.

While resolving convex/concave ambiguity is generally known to be impossible to solve properly, we find clues about stroke labeling patterns at local regions: locally proximate strokes often have correlated convexity properties. Based on the correlations, we solve the question by regarding the strokes as a whole and searching for the most consistent stroke labeling.

Before introducing the correlations, first we define the notion of *simple strokes*, which the correlations are built upon, as strokes that only bend in one direction in the 2D plane. Formally, given a stroke  $s(t), t \in [0, 1]$ , the signed curvature  $k(t) = \frac{\det(s'(t), s''(t))}{\|s'(t)\|^3}$ ; if  $k \geq 0$  (or  $k \leq 0$ ), the stroke is a simple one. Notice that when the stroke is parametrized reversely,  $k$  changes sign as well. Thus when we choose the traversal tangent direction  $\vec{T}$  of a simple stroke, we can say the stroke is planar convex ( $k \geq 0$ ) or concave ( $k \leq 0$ ), which we denote as  $\text{sign}(s, \vec{T}) \in \{+, -\}$ .

We make the following observations about the convex/concave properties of locally correlated bending strokes (Fig. 3):

- a simple stroke is slightly more likely to be convex (the convex prior);
- two sequential simple strokes with the same/opposite planar convexity are likely to have the same/opposite convexity properties;
- two parallel simple strokes with the same planar convexity are likely to have the same convexity property.

Note that in (b) and (c) when we compare two strokes, we choose their traversal tangent directions to have an angle less than 90 degrees. The convex prior (a) is a reasonable default assumption, since we are likely to see the convex parts of an object while the concave parts can be hidden; such a prior is also used in 3D reconstruction methods in computer vision [Barron and Malik 2015].

To analyze the convexity of strokes, the input strokes are first segmented into simple strokes, and then a graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is built, with each simple stroke as a node in  $\mathcal{V}$ , and an edge in  $\mathcal{E}$  between two nodes if the corresponding strokes fall into the situation of (b) or (c). Then we solve the following standard graph labeling problem:

$$\min .E(x) = \sum_{s \in \mathcal{V}} \theta(x_s) + \sum_{(s, t) \in \mathcal{E}} \theta(x_s, x_t), \quad (1)$$

where the labeling  $x_s \in \{0, 1\}$  denotes if the stroke  $s$  is convex or concave. Following (a), for the unary function, we set  $\theta(0) = 0.4$  and  $\theta(1) = 0.6$ , favoring convex strokes. For the binary function, we set  $\theta(0, 0) = \theta(1, 1) = 0.0$  and  $\theta(1, 0) = \theta(0, 1) = 1.0$  if the two strokes are supposed to have the same convexity, and use  $1 - \theta(\cdot, \cdot)$  if they are supposed to be opposite. Finally, the user can specify the convexity of selected strokes, which are constraints to the above labeling problem for the other strokes, as a way to fix the errors made by this automatic labeling algorithm.

To solve the graph labeling problem, we can either do an exhaustive search of the combinatorial space of all possible labelings, or use an approximate solver for efficiency. In our implementation, we use the algorithm by [Kolmogorov 2006]; because the number of strokes per view is quite small, usually in the range of 10 ~ 40, the heuristic algorithm works very well in finding nearly optimal solutions. Fig. 2 shows an example where long strokes were segmented into simple strokes representing opposite surface curvatures. See Sec. 5.1 for more examples.

### 3.3 Domain triangulation

For efficient and flexible surface construction, we triangulate the 2D domain  $\Omega$  defined by the boundary curves  $\Gamma$  into a planar mesh  $M$ , and the height field surface is discretized by the lifted version of the triangle mesh  $M$ , with the variables encoded by the z-coordinates of the mesh vertices. We use the Delaunay refinement algorithm of CGAL [CGA 2016] to quickly generate a dense and high quality mesh confined to the boundary curves. Note that if there are feature curves sketched, they should be preserved by the planar mesh, so that there are mesh edges located on the feature curves to represent sharp features in the surface (Fig. 2(b)).

To adapt the triangle size of the planar mesh to different domain shapes, we resample the boundary curve according to the local feature size function [Amenta and Bern 1998] defined over the boundary, and the Delaunay refinement algorithm with edge length and angle quality criteria naturally induces smooth gradation of triangles inside the domain (Fig. 2(b)).

### 3.4 Surface from curvature field

Because the bending strokes are approximate curvature lines of the spatial surface projected on plane, we use this cue to first recover the dense curvature field of the surface as parameterized over the 2D region, and then reconstruct the surface matching the curvature field.

**Problem formulation.** Let the spatial surface be parameterized as a Monge patch over the 2D domain, that is,  $z = f(x, y), (x, y) \in \Omega$ . The Weingarten shape operator  $dN : \mathbf{w} \rightarrow \mathbf{w}', \mathbf{w}, \mathbf{w}' \in \mathbb{R}^2$  is a linear mapping that encodes both the principal curvature directions

and magnitudes, denoted as  $\{\mathbf{u}, \mathbf{v}, \lambda_u, \lambda_v\}$ , as its two eigenvectors and corresponding eigenvalues. Meanwhile, the shape operator can be deduced from  $dN = -\text{II}G^{-1}$ , where  $G$  and  $\text{II}$  are the first and second fundamental forms of the surface  $z = f(x, y)$ . As such, given a curvature field, we can infer the surface shape through minimizing the following energy:

$$E_{match}(z) \equiv \frac{1}{|\Omega|} \int_{\Omega} \|dN \cdot \mathbf{u} - \lambda_u \mathbf{u}\|^2 + \|dN \cdot \mathbf{v} - \lambda_v \mathbf{v}\|^2 d\sigma. \quad (2)$$

For numerical computation, the integrand is discretized as piecewise constant over the triangle mesh  $M$ , and the energy is a summation of the integrand for each triangle while weighted by triangle area. In the Appendix, we show how to discretize the shape operator on a triangle mesh, as well as how to optimize this nonlinear functional through an iterative process, where in each iteration we solve a simple linear problem, thus making the whole process computationally efficient.

**Remark.** If there are isophotes sketched [Xu et al. 2015], they can be naturally incorporated into the above problem: suppose we know at a point the lighting direction  $\mathbf{l}$ , and the isophote direction  $\mathbf{u}$ , they form conjugate directions with respect to the shape operator, thus we have  $\mathbf{l}^T dN \mathbf{u} = 0$ .

Next we discuss the computation of the curvature field given input bending strokes. The curvature direction field  $\{\mathbf{u}, \mathbf{v}\}$  is defined as a piecewise constant non-orthogonal 4-direction field over  $M$ , and the curvature magnitude field  $\{\lambda_u, \lambda_v\}$  is defined similarly as two piecewise constant scalar fields.

*Curvature direction field.* We adopt the formulation of BendField energy [Iarussi et al. 2015] to regulate the principal curvature directions of a smooth surface as projected on a plane:

$$Ebend\_field(\mathbf{u}, \mathbf{v}) \equiv \frac{1}{|\Omega|} \int_{\Omega} \|\nabla_{\mathbf{u}} \mathbf{v}\|^2 + \|\nabla_{\mathbf{v}} \mathbf{u}\|^2 d\sigma, \quad (3)$$

subject to  $\mathbf{u}, \mathbf{v}$  following bending stroke directions where available. Here  $\nabla_{\mathbf{v}} \cdot$  is the covariant derivative operator; in the Appendix we show how it is discretized on a triangle mesh. Note that the BendField energy is a bi-convex functional of the variables.

To minimize the energy under directional constraints of the bending strokes, following [Iarussi et al. 2015], we initialize  $\mathbf{u}, \mathbf{v}$  as a harmonic non-orthogonal 4-direction field. The directions are further refined through an iterative sequence of linear optimizations, where in the  $k$ -th step the new  $\mathbf{u}_k, \mathbf{v}_k$  are solved as

$$\operatorname{argmin}_{\mathbf{v}} \frac{1}{|\Omega|} \int_{\Omega} \|\nabla_{\mathbf{u}_{k-1}} \mathbf{v}\|^2 + \|\nabla_{\mathbf{v}_{k-1}} \mathbf{u}\|^2 d\sigma.$$

In all our experiments, we find 5 iterations are sufficient for convergence, thus the optimization is very efficient.

Compared with [Iarussi et al. 2015], we gain large efficiency improvement due to two factors: first, our discretization is on the triangle mesh which is much more sparse than the pixel grid; second, we solve the harmonic non-orthogonal 4-direction field through a new approach specified in the PolyVector representation [Diamanti et al. 2014], details of which is presented in the Appendix.

*Curvature magnitude field.* Because the principal curvature values follow a similar variation to the curvature directions, we regulate

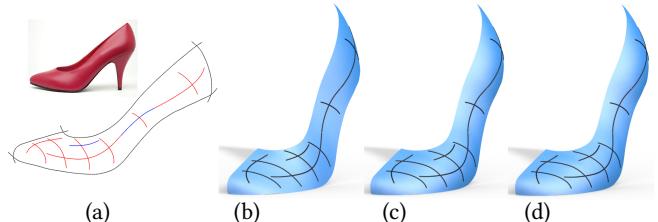


Fig. 4. The optimization process of a shoe model patch. (a) is the reference image and the planar sketch, with bending curves parsed into convex/concave simple strokes. (b) is the surface after the initialization step. (c)&(d) show surfaces after 3 and 5 iterations. The planar curves are projected to the surfaces for visualization.

the principal curvature values through minimizing the following energy:

$$E_{\lambda}(\lambda_u, \lambda_v) \equiv \frac{1}{|\Omega|} \int_{\Omega} \|\nabla_{\mathbf{u}} \lambda_v\|^2 + \|\nabla_{\mathbf{v}} \lambda_u\|^2 + \beta (\|\nabla_{\mathbf{u}} \lambda_u\|^2 + \|\nabla_{\mathbf{v}} \lambda_v\|^2) d\sigma, \quad (4)$$

subject to the estimated surface curvature values along the bending strokes. Here  $\nabla_{\mathbf{u}} \cdot$  is now applied to scalar functions, thus becoming the common directional derivative operator, and  $\beta > 0$  is the weight of the curvature variation term [Joshi and Séquin 2007]. In our experiments, we set  $\beta = 0.01$  to induce a weak smooth variation of curvatures which leads to surface fairness. Note that since the curvature directions  $\mathbf{u}, \mathbf{v}$  are not variables, the energy is a quadratic functional of the curvature values which can be easily solved.

Until now, while the principal curvature directions can be solved, we do not know how much the surface bends along a stroke; without the curvature values along bending strokes as constraints, the curvature magnitude field cannot be recovered through minimizing the above energy  $E_{\lambda}$ . In the next section, we discuss how to compute the curvature magnitudes and the spatial surface simultaneously.

### 3.5 Joint optimization of surface and curvature magnitudes

We solve the height field surface and curvature magnitudes through two major steps: first, given the convex/concave labeling of the bending strokes, we make an initial guess of the surface shape by assigning a uniform curvature value to the bending strokes; second, starting with the initial guess, we iteratively update both the curvature values along the strokes and the surface shape to make them consistent.

*Initialization.* We find an initial estimation of the surface first, so that subsequent joint optimization of surface shape and curvature field can escape the trivial solution of a flat surface with zero curvature everywhere that minimizes  $E_{match}$ .

For each simple bending stroke with labeled convexity denoted as sign  $\pm$ , we assign to the whole stroke a constant absolute spatial curvature value  $\mu > 0$ , thus its corresponding principal curvature is  $\pm\mu$ . Then we minimize the energies  $E_{bend\_field}$  and  $E_{\lambda}$  under constraints of the bending strokes, to get the complete curvature field, with which  $E_{match}$  is minimized to find an initial guess of the spatial surface. Since we only need an initial surface estimation, the nonlinear functional  $E_{match}$  is not fully minimized in this step; in our experiments, only one iteration of optimization is carried out.

Note that while the value  $\mu$  would certainly affect the surface shape at this initialization step, random choices of  $\mu$  from a reasonable range lead to final surfaces with no noticeable difference. In our experiments we simply set  $\mu = 1$ .

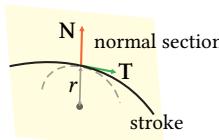
*Joint optimization.* The shape of a planar bending stroke is affected by two factors, namely the spatial shape of the stroke as embedded in the surface and the surface orientation with respect to the projection direction. Thus given the planar stroke, it is impossible to recover the true surface curvature along the stroke without knowing the surface first, which is why we devise a joint optimization to solve the surface curvature along a stroke and the surface simultaneously.

The joint optimization is done through an iterative process, with each iteration consisting of the following three steps:

- (1) estimate principal curvatures by analyzing the lifted strokes;
- (2) diffuse curvature values through minimizing  $E_\lambda$ ;
- (3) update surface shape by minimizing  $E_{match}$  for one iteration.

Note that in the process the curvature directions are unaltered; as a result, the quadratic energy  $E_\lambda$  only has its constraints (i.e. curvature values along bending stroke) varied between iterations, and can be efficiently optimized by factoring its normal equations once and using back substitution for solving updated curvature values in subsequent iterations.

*Estimating principal curvatures.* To estimate the principal curvature along a bending curve, we lift the planar curve to the current spatial surface. For each point on the lifted curve (see inset), the normal section plane is spanned by the surface normal  $N$  at the point and the curve tangent  $T$ , on which we project a neighborhood of the spatial curve, and fit a circle to the projected image; the reciprocal of circle radius  $r$  is the estimated principal curvature value along the curve direction at the point. The sign of curvature is solely determined by bending stroke convexity labeled before.



The neighborhood size to estimate curvature with should be chosen: if the neighborhood is too big, the estimated curvature can be too conservative and inflexible, while a size too small causes instability and inaccuracy due to the locally noisy shape of the lifted stroke. In our experiments, we find a curve segment of length  $D/30$  allows both flexibility and robustness, where  $D$  is the diagonal length of the domain bounding box. For fast computation, we simply divide a lifted stroke into segments of approximate length  $D/30$ , and estimate the curvature for each segment, which is then assigned to stroke points belonging to the segment. To carry out the circle fitting for curvature estimation, we minimize the standard geometric fitting error, plus a regularization of radius for robustness:

$$\min_{c,r} \frac{1}{N} \sum_{i=1}^N (\|p_i - c\| - r)^2 + \mu \cdot r^2,$$

where  $\{p_i\}$  are the points to be fitted,  $(c, r)$  define the circle, and  $\mu = 2 \times 10^{-5}$  is a very small weight. This optimization is a nonlinear least square problem with only 3 variables and is solved by Gauss-Newton iterations.

*Convergence.* The joint optimization stops when the surface update in step (3) is below a certain threshold. For all examples shown, we find 5 iterations are sufficient for such a convergence.

Note that the input sketches may correspond to more than one possible shape, in which case the above iterations do not necessarily converge to the desired one, or may not converge at all. To resolve this problem and get meaningful shapes, we need to use additional boundary conditions to constrain the solution space. In Section 3.6 we discuss commonly used boundary conditions.

*User adjustment.* While the above scheme for simultaneous estimation of principal curvature along strokes and surface generally finds the proper pattern of shape variation, it is possible that the surface differs from the user expectation, which can be attributed to two factors: 1) that the strokes are sketched with insufficient accuracy, and 2) the bas-relief ambiguity [Belhumeur et al. 1999] which says that differently scaled height-field surfaces have similar 2D images. Thus the user may want to modify the strokes and adjust the solved shape to better match sketch intention.

To facilitate the adjustment, our system allows the modification of sketched strokes through curve editing, the scaling of curvatures along selected parts of strokes, and the scaling of the whole curvature field. Once the shape of a sketched curve is modified, the whole process of single-view modeling is recomputed to update the surface. When the curvatures along selected parts of strokes are scaled (c.f. Fig. 12), the curvature magnitude field is recomputed by minimizing  $E_\lambda$  under new constraints, and the surface is updated by fully optimizing  $E_{match}$ . If the user finds the overall shape can be more curvy or otherwise, the curvature magnitudes are uniformly scaled and the surface updated by minimizing  $E_{match}$  fully.

### 3.6 Boundary conditions and other strokes

*Boundary conditions.* The bending strokes and the curvature field derived specify differential properties of the surface. To determine a unique surface, we still need sufficient boundary constraints. We impose the following constraints on the height values along boundary curves  $\Gamma$ :

- positional constraint - all boundary points should be close to the drawing plane or a specified boundary curve  $z_0(p)$ , which amounts to penalizing  $\frac{\omega_0}{|\Gamma|} \int_{p \in \Gamma} (z(p) - z_0(p))^2 dp$ , where  $z_0(p) = 0$  for planar boundary,
- regularity constraint - the boundary curve should be smooth in space, by penalizing the variation of heights  $\frac{\omega_1}{|\Gamma|} \int_{p \in \Gamma} z''(p)^2 dp$ .

Here  $\omega_0, \omega_1 > 0$  are weights,  $|\Gamma|$  is the length of boundary curve. The sum of the two terms denoted  $E_{bdry}$ , a quadratic function of  $z$ , is added to  $E_{match}$  when solving for the height field surface. For many examples we wish the boundary curve to be exactly as prescribed, and a big  $\omega_0$  is used; on other cases where we look for more flexible boundary curves, a decreased value for  $\omega_0$  is adopted. For example, the container (Fig. 1) has  $\omega_0 = 100$ , the leaf model (Fig. 2)  $\omega_0 = 10$ , and the outer boundary of hat and shell (Fig. 6) have  $\omega_0 = 100$  and 0.1 respectively, to allow the natural formation of curvy boundaries; for the rest of the examples, we simply use a big  $\omega_0 = 10^6$  to stick to the given boundaries. We have used  $\omega_1 = 1$  for all the test cases.

*Handling the other strokes.* Strokes other than bending curves have their specific properties to impose on the constructed surface (Sec. 3.2.1). Here we discuss in detail how they are incorporated when solving for the surface.

Contour curves provide normal vector constraints on the surface at the boundary. Specifically, the normal vector of the contour curve in plane (see inset,  $N_{bdry}$ ) should also be orthogonal to the surface tangent plane there. However, given that the border triangle  $f$  has a non-vanishing planar projection, it can never be orthogonal to  $N_{bdry}$ . So in practice, we require the normal of the border triangle to be an interpolation of the boundary normal  $N_{bdry}$  and the surface normal  $N_v$  at the interior vertex of the triangle, formally  $\nabla z = g$ , where the interpolated normal vector is converted to gradient  $g$  such that  $(g_x, g_y, -1) \parallel (2N_v + N_{bdry})$ . The interpolation ratio of 2 : 1 is an empirical value that we find balances the contour normal constraint and surface regularity, as a ratio too large may lead to border faces in plane, while a ratio too small can cause distorted border triangles with extreme heights. The contour normal constraint thus is denoted  $E_{cntr} = \frac{1}{\sum_f |f|} \sum_f |f| \| \nabla z_f - g_f \|^2$  which is summed over contour-border triangles.

If for certain cases, the user wants a strict perpendicularity between contour normals and the surface, we allow postprocessing of the computed surface to meet the requirement, by a shape preserving Laplacian deformation [Sorkine et al. 2004] of the surface plus enforcing the contour normal constraint and penalizing point-wise distance from the original surface mesh.

The ridge/valley curves are converted to a set of bending lines with known convex/concave labels (see inset). The generated bending strokes are sampled along the ridge/valley curves, and orthogonal to them. They are then included as input to the above mentioned surface computation algorithm. In our implementation, the distance between consecutive generated bending strokes is around  $D/10$ , which is also the length of a generated bending stroke, where  $D$  is the bounding box diagonal length of the planar sketching. An example shape modeled with many ridge and valley curves is shown in Fig. 6.

A flat stroke imposes a constraint on the normal vectors of mesh faces crossed by the stroke, requiring the normal vectors to be parallel. This condition is translated into additionally penalizing the variation  $\|\nabla z_i - \nabla z_{i+1}\|^2$  when solving surface, where  $i$  indexes the sequence of  $N$  faces crossed, and  $\nabla z$  the gradient of the height function on a mesh face. We denote this term  $E_{flat} = \frac{\omega}{N} \sum_{i=1}^N \|\nabla z_i - \nabla z_{i+1}\|^2$ , with weight  $\omega = 10^3$  in our experiments.

A sharp feature, on the other hand, induces a sharp angle between two neighbor triangles across it. Thus it is implemented by prescribing the difference between height field gradients on the triangle pairs  $i, j$  crossing it:  $\|\nabla z_i - \nabla z_j - s e_{ij}^\perp\|^2$ , where  $e_{ij}^\perp \in \mathbb{R}^2$  is the direction vector of the shared edge between  $i, j$  rotated by 90°, and  $s \in \mathbb{R}$  is the target difference magnitude. The penalization is denoted  $E_{fea} = \frac{1}{|L|} \sum_{i,j} |\epsilon_{ij}| \cdot \|\nabla z_i - \nabla z_j - s e_{ij}^\perp\|^2$ , with  $|L|$  the total length of sharp feature curves.

Note that  $\nabla z_i - \nabla z_j$  measures the isotropic curvature [Koenderink and van Doorn 2002; Pottmann and Liu 2007] across the shared edge,

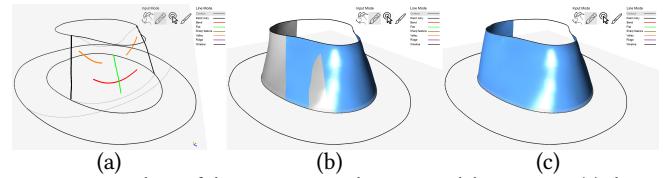
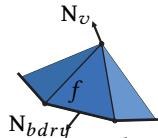


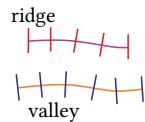
Fig. 5. Screen shots of the prototype multi-view modeling system. (a) shows the sketched strokes for the front patch, (b) shows the computed surface (blue) along with a previously created side patch (gray), and (c) the two patches are merged tightly. A complete model is shown in Fig. 6.

which approximates the Euclidean curvature of the surface but is different. We adopt such a measurement because of its simplicity as a linear function of the height variables, and its known direction along  $e_{ij}^\perp$  which has been used in other situations like modeling discrete self-supporting surfaces [Liu et al. 2013; Vouga et al. 2012] where the convexity of the lifted surface is critical.

To summarize, the complete energy for solving the height surface is  $E_{match} + E_{bdry} + E_{cntr} + E_{flat} + E_{fea}$ , of which all but  $E_{match}$  are quadratic functionals. Thus the computational complexity for solving the surface does not change.

#### 4 MULTIPLE VIEW MODELING SYSTEM

In this section we consider the modeling of more complex 3D objects which cannot be sketched from a single view due to occlusions. For many symmetric objects, we may sketch one side and simply mirror the patch to the other side to form the complete shape, like many examples shown in this paper (Figs. 1, 9, 13). For other objects, surface patches for different views should be sketched separately and merged. We present a multi-view modeling process that supports the flexible change of views for sketching and the seamless combination of sketched patches. A screen shot of our prototype modeling system is shown in Fig. 5. The supplemental video shows complete sessions of using the system to create various shapes.



*System overview.* The building blocks of our system are a set of drawing planes positioned appropriately in the 3D space, over which the user sketches and the height field surfaces are constructed.

To assist the drawing of objects with complex boundary curves, our system also supports the modeling of spatial boundary curves through sketching. We implement the curve modeling as a spline fitting procedure, by converting a boundary curve to B-spline curve and fitting it to user-drawn curves on specified drawing planes. The curves can also be modeled by more sophisticated methods (e.g. [Bae et al. 2008]) and imported to our system.

Finally given the surface patches corresponding to multiple views, we deform the surface patches in a shape-preserving way and seamlessly merges them into complete objects.

*Drawing plane.* The sketched strokes are mapped to a drawing plane through inverse perspective projection: for each point of a stroke, a ray starts from the camera, passes through the point in the image plane and hits the drawing plane at the mapped stroke point. Drawing planes can be created where necessary, and can be modified by rigid transformations to more appropriate position and orientation for sketching and modeling.

*Merging surface patches.* As a new surface patch of a conceived 3D shape is sketched, it should be registered and merged with the other

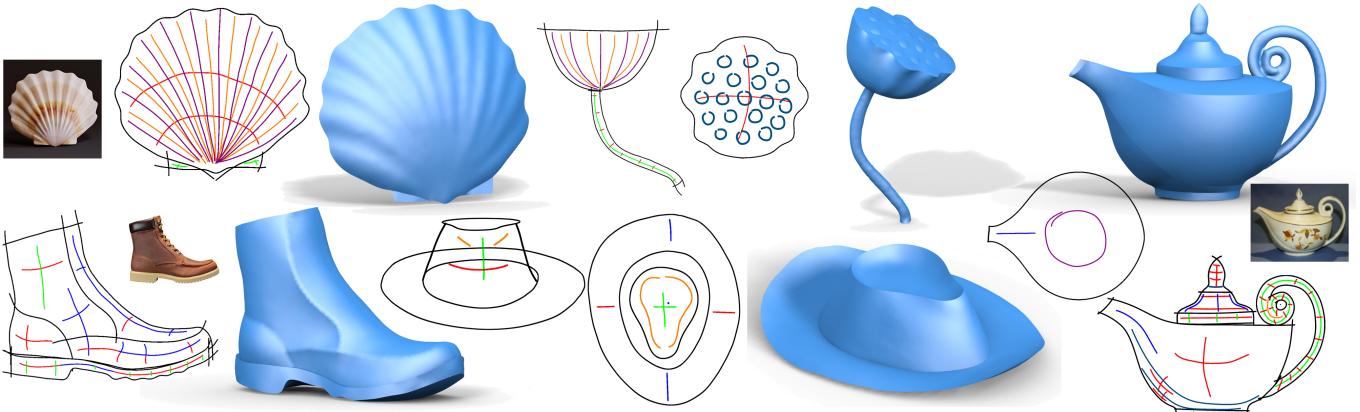


Fig. 6. Five more shapes created with our method. The shell is a single surface patch, and the others are made of multiple view sketching. The shell, boot and teapot models are sketched by referring to the real world images, while the lotus seed pod and hat are created freely without specific references. Due to space limit, the sketches for the bottom of teapot, the back side and bottom of boot, and the two side views of hat, are not shown here.

surface patches. To find the correspondence with the other patches, we simply project the other patches in 3D onto the current drawing plane through orthogonal projection  $\Pi$ , and a correspondence is formed between two spatial points  $p$  of the current patch and  $q$  of the rest patches whenever their projected points  $\Pi(p) = \Pi(q)$ . Thus for each vertex of the rest patches, we find its corresponding point in the current patch if exists, and put the pair into a set  $C$ .

To merge the patches, we eliminate the gap in-between through a shape preserving deformation of the current patch and its neighbor patches with established correspondences, by solving the optimization problem:  $\min_{\{x\}} E_{laplace} + wE_{corr}$ , where  $E_{laplace}$  is the Laplacian surface deformation energy [Sorkine et al. 2004], and

$$E_{corr} = \sum_{i \in C} \|p_i - q_i\|^2 + \|n_i \cdot (p_i - q_i)\|^2,$$

with  $p_i = \alpha x_{i0} + \beta x_{i1} + (1 - \alpha - \beta)x_{i2}$  represented in barycentric coordinates  $\alpha, \beta$  of the vertices  $x_{i0}, x_{i1}, x_{i2}$  of the triangle containing  $p_i$ ,  $n_i$  the normal vector of the vertex  $q_i$ .  $w = 10^5$  is a large weight to force merging of the patches tightly.

In Fig. 5 we see a hat model is being created through multiple views, with a new patch sketched and merged with previously created patches. Note that the spatial wireframe is modeled beforehand as boundary curves to anchor the individual patches.

## 5 RESULTS AND DISCUSSION

In Figs. 1 and 6 we show more freeform 3D shapes created through our approach, which contain both single surface patches and complex models made of multi-view sketching. Still more examples are included in the supplemental materials.

With our prototype system running on a desktop PC with a CPU of 4 cores and 3.4GHz, the run time for a typical sketch with 20 strokes, and a domain triangulation with 7k faces, takes about 80ms for stroke parsing and meshing, 1200ms for curvature direction field computation, and 1700ms for solving the height surface, thus around 3s in total. Table 1 lists the runtime breakdown of concrete examples. Note that while here we measured the total execution time for each example, in the typical scenario, the user incrementally inputs more strokes to explore and refine the models, which suggests that for

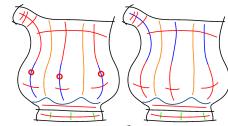
Example	#face/vert	#stroke(#flip)	meshing	bend field	surfacing
Crane Fig. 1	7.4k/4.0k	28(0)	78	1202	1664
Pillow Fig. 1	8.4k/4.4k	54(0)	91	1651	1997
Teapot Fig. 1	8.6k/4.5k	24(3)	82	1727	2218
Container Fig. 1	8.2k/4.3k	17(2)	75	1744	2083
Skirt(front) Fig. 1	6.5k/3.4k	35(1)	67	1289	1661
Skirt(back) Fig. 1	5.9k/3.1k	24(0)	81	1154	1516
Leaf Fig. 2	8.0k/4.2k	22(1)	110	1303	1809
Shell Fig. 6	8.7k/4.5k	33(0)	64	1858	2542
Boot Fig. 6	5.8k/3.0k	21(5)	77	955	1342

Table 1. Runtime breakdown of several examples, measured in ms. #flip is the number of user fixed labels of stroke convexity. For examples with multiple patches, the data is for the patches with shown sketches. The convexity labeling step (not shown here) always takes less than 1ms.

faster feedback and smoother interaction, we can use a simple multi-resolution strategy: we compute a preview of the updated surface with a lower resolution mesh quickly, and replace it with a finer mesh of more details when ready. Indeed, the time cost has been measured to be roughly proportional to the mesh size.

### 5.1 Algorithm validation

*Convexity labeling.* The automatic convexity labeling of bending strokes is effective in many cases (see Figs. 2, 4, 9, Table 1). However, there are cases when the user needs to do manual fixations. One such situation is when the strokes do not have local proximity and thus the rules for automatic labeling cannot apply; as a consequence the strokes may simply be labeled as convex and the user should fix it when necessary (see Boot (Fig. 6) for example). Another situation is that the automatic labeling of strokes happens to be the opposite of expectation; this is due to the bas-relief ambiguity which requires user fixation. A labeling fixation example is shown in inset, where the user has to flip the labeling of three strokes (marked by red circle) and the rest are automatically updated.



*Comparison with ground truth shapes.* We compare the surface computed from a planar sketching consisting of accurate projected curvature lines of a torus patch against the ground truth surface

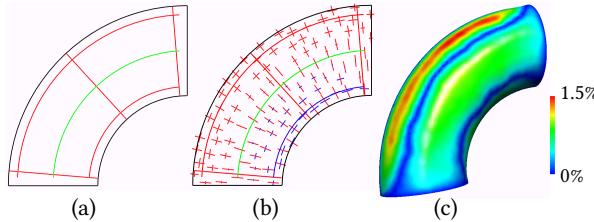


Fig. 7. Comparison with a 1/8 torus. (a) the input planar sketch. (b) the curvature field. (c) the solved surface color-coded by the distance from ground truth. The maximum error is 1.5% of bounding box diagonal length, and the average error is 0.5%. The correct boundary curve is prescribed.

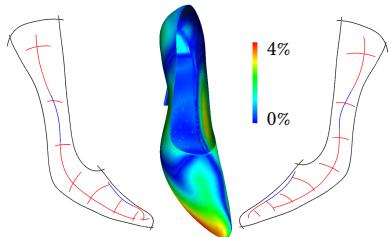


Fig. 8. Compare a sketched shoe surface (formed with two patches whose sketches are shown on both sides) with known ground truth 3D model. The result surface is color-coded by its difference from ground truth. The maximum error is 4% of bounding box diagonal length, and average error 0.7%. The correct boundary curve is prescribed.

(Fig. 7). The error from ground truth is very small, proving that the iterative surface optimization from planar strokes is effective and accurate. We also take a known 3D shoe model and try to recover the model by sketching. As shown in Fig. 8, a sparse number of strokes drawn manually already lead to a 3D model very close to the ground truth. Note that for the two comparisons, the boundary curves are prescribed according to ground truth data.

## 5.2 User evaluation

The proposed sketch-based modeling approach is also evaluated for its effectiveness and intuitiveness by novice users. We have invited nine participants to learn and use the method to create freeform shapes. To teach the users how to sketch shapes with bending strokes and other strokes, we show them three examples with target images, the strokes we sketched and the corresponding 3D models; they can also try to practice their understanding of the method with some of the given examples. Such a training session usually takes 30 mins. After that, each participant sketches three target shapes which are depicted through images. We find that the participants generally created satisfying shapes within an average period of 10 mins for each target shape. Some of the training examples and user created testing models are shown in Fig. 9. Finally, we ask the participants to rate their experiences with the sketch-based modeling method and how intuitive and effective it is. The overall rating is again quite positive. The details of the user evaluation are included in the supplemental materials.

Among the nine participants, two are artists who work on a daily basis with 3D modeling suites Autodesk Maya and SolidWorks, respectively. They find that the new approach is very quick to learn, especially compared with the 3D modeling suites they use, and significantly shortens the time required to create draft 3D shapes

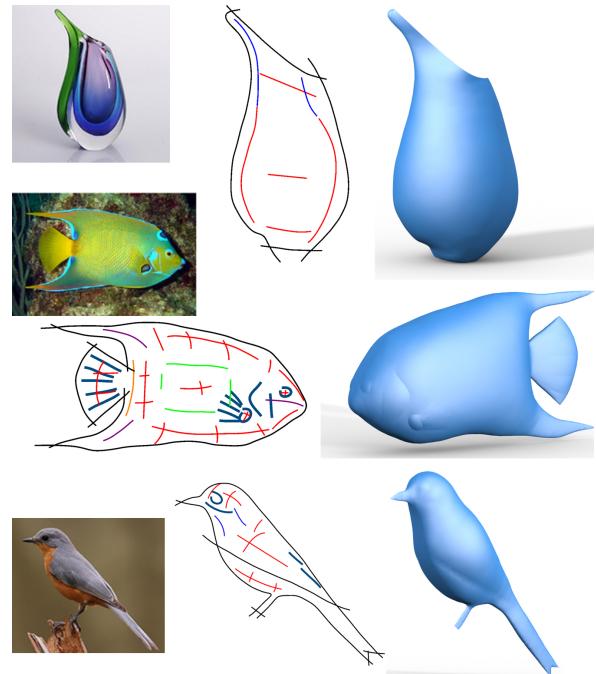


Fig. 9. Training and test examples used for user evaluation. The first row is an example used for training the users. The other two rows show examples sketched by different users.

which can be further edited for details in other software, with an estimated saving of 2/3 of time cost with their commonly used tools.

## 5.3 Comparisons

*BendField* [Iarussi et al. 2015]. In [Iarussi et al. 2015], the bending strokes are used to derive a projected curvature direction field, and the normal vectors for 3D visualization are recovered by transforming the two principal curvature directions  $\mathbf{u}, \mathbf{v} \in \mathbb{R}^2$  at each point into orthogonal 3D vectors  $\vec{U} \perp \vec{V}, \vec{U}, \vec{V} \in \mathbb{R}^3$  while striving for direction smoothness, exploiting the fact that the principal directions of a spatial surface should be orthogonal except for umbilical points. Such an approach recovers smooth normal vectors good for rendering a 3D impression for more regular shapes which are formed of surface patches with relatively uniform normals; however, as we show next through examples (Figs. 10&11), it fails to find proper normal estimations for more freeform shapes, which further prohibits the next step of surface reconstruction.

In contrast, in our approach we model a complete curvature tensor field with both curvature directions and magnitudes, and use it to drive the reconstruction of spatial surface directly, rather than through normal vectors. As a result, our approach enables a robust surface reconstruction algorithm, as is demonstrated through the examples.

In this comparison, to build a surface from normal vectors for the BendField method, we take the standard approach of integrating normal vectors to form surfaces, by solving:

$$\min . \sum_{i \in \text{faces}} \|\mathbf{n}_i \cdot (\mathbf{p}_{i1} - \mathbf{p}_{i0})\|^2 + \|\mathbf{n}_i \cdot (\mathbf{p}_{i2} - \mathbf{p}_{i0})\|^2 + \gamma \sum_{j \in \text{verts}} \|\Delta \mathbf{p}_j\|^2, \quad (5)$$

where  $\mathbf{n}_i$  is the normal vector for the  $i$ -th triangle face,  $\mathbf{p}_{i0}, \mathbf{p}_{i1}, \mathbf{p}_{i2} \in \mathbb{R}^3$  its vertices, and  $\Delta p_j$  the Laplacian of the  $j$ -th vertex to measure surface smoothness. We use a coefficient  $\gamma = 0.1$  to regulate the surface reconstruction. The boundary conditions are simply to fix the boundary vertices in plane for the two examples shown.

By observing the first test case of a pear shape, we find the curvature direction fields of ours and BendField are almost the same, but the reconstructed normal vectors are largely different: the normal vectors of BendField result do not capture the proper shape variation. The BendField method allows the user to manually fix the normal orientation errors of regions, which is to click on a point that belongs to the wrongly oriented region and propagate the flipped orientation to the entire region. We tried to fix such orientation errors in the initial normal estimation, as shown in Fig. 10(g); as we see in Fig. 10(h), after the manual fixation the recovered normals show the variation of the shape more clearly. However, the new normals are not accurate enough for reconstructing a proper surface (Fig. 10(i)). Note that we tried to flip normals which seem to be obviously wrongly oriented, and there can be other corrections, but it is unclear if any other correction will lead to a better normal estimation or surface reconstruction.

In the second test case of a more freeform bird model, we see the normal estimations by BendField do not capture the shape variations (Fig. 11(e)), and the fixation of normal orientations does not erase this problem completely (Fig. 11(h)). In addition to normal and surface reconstruction, this example further shows how the curvature direction field by our method differs from that by BendField method, especially around the neck and head of the bird model. It demonstrates that our curvature direction computation method produces smooth direction fields with better alignment to the sketched strokes than BendField, which can be attributed to the adoption of a convex harmonic 4-direction field formulation as well as the reduced number of variables in a triangle mesh discretization, compared with the nonconvex mixed-integer formulation and a dense pixel grid for the BendField approach.

**True2Form** [Xu et al. 2014]. Our approach is largely different from [Xu et al. 2014], where a majority of planar curves are also regarded as projected curvature lines but the lifting of the 2D curve network to 3D is based on regularities of the curve network for many man-made objects. In comparison, here we do not assume the sketches form a connected and nicely laid out curve network, which is more suitable for modeling freeform shapes with complex curvature patterns. On the other hand, while True2Form is able to convert multiple-layered curve networks into 3D, our method models each patch as a height field and cannot handle multiple layers.

**Shape from shading.** We compare our method with the state-of-the-art shape from shading method [Barron and Malik 2015]. As shown in Fig. 12, the shape-from-shading reconstruction as a well-known ill-posed problem is frequently incapable of recovering quality shapes even if there are advanced priors and simultaneous estimation of lighting, reflectance and geometry.

**Biharmonic surfaces.** Many contour-based modeling methods build smooth shapes by solving the biharmonic equation  $\Delta^2 \mathbf{x} = 0$  under various

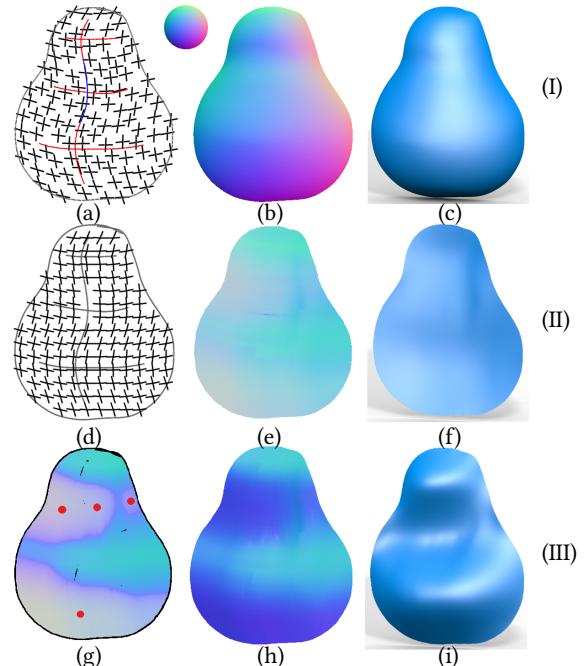


Fig. 10. Comparison with the BendField method [larussi et al. 2015] through a pear shape. (I) are results of our method, (II) of BendField, and (III) of BendField with normal orientation editing. (a)&(d) are the curvature direction fields. (b),(e),(g),(h) are normal maps, with (e)&(h) the results of BendField, and (g) showing normal flipping operations (at red dots). (c),(f),(i) show reconstructed surfaces.

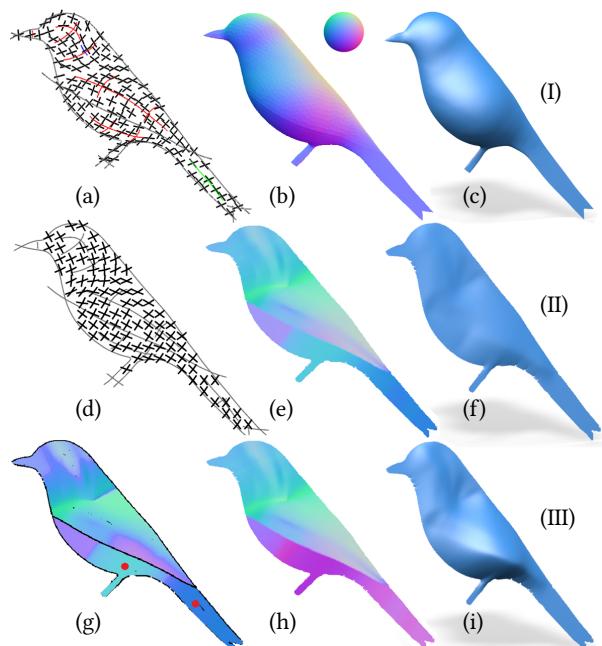


Fig. 11. Comparison with the BendField method [larussi et al. 2015] through a bird shape. See caption of Fig. 10 for explanation. Note how the direction fields differ in this case, and our result better aligns with the sketched curves.

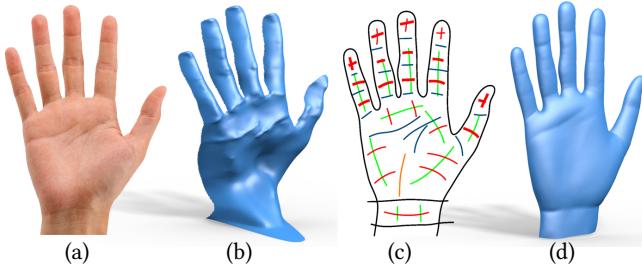


Fig. 12. Comparison with a shape-from-shading reconstruction method [Barron and Malik 2015]. Objects with complex shading/material generally fails automatic SFS methods (b). With our approach, a sketching with few curves (c) can create a quality 3D model (d). The different widths of bending strokes in (c) show the user adjustment of stroke curvature scaling (Sec. 3.5).

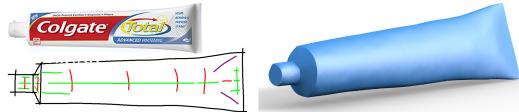


Fig. 13. A failure case for 3Sweep [Chen et al. 2013] is easily created by our method. The 3D model is made of a patch from the sketching and its symmetric copy.

constraints [Joshi and Carr 2008; Nealen et al. 2007; Yeh et al. 2016]. While flexible, the biharmonic equation does not allow directional control of the geometry, and excludes anisotropic shapes like torus as solutions. On the other hand, in our approach the modeling of the complete curvature tensor field avoids this limitation. In inset we show the biharmonic surface given the boundary conditions of a torus patch, which largely differs from the ground truth; the user may manually specify a larger mean curvature to the interior to further inflate the surface, but it is difficult and unintuitive to assign accurate curvature values. Note that a similar limitation with biharmonic surfaces is also discussed in [Pan et al. 2015] (Fig. 10), where an undesirable isotropic surface is compared with the result of [Pan et al. 2015] which also has directional control through curvature tensor modeling.

*Other methods.* As is mentioned before, many previous methods target specific classes of geometric objects. Our method can thus be a useful complement to the existing works. For example, a failure case of 3Sweep [Chen et al. 2013] can be easily modeled by our method (Fig. 13).

## 6 CONCLUSION

We have presented a sketch-based modeling method that utilizes the bending strokes primarily to model freeform shapes with complex curvature patterns. The approach has been demonstrated to be intuitive to users, and allows the quick creation of a large variety of interesting shapes. To find surface from the bending strokes, technically we solve several key problems, including the disambiguation of bending strokes representing convex/concave curvature, the estimation of bending magnitudes along the strokes, and the construction of surface from the sparse bending strokes through modeling a smooth curvature field conforming to the strokes. To handle complex objects with self-occlusions, we extend to multiple views, where

surface patches for all the views are sketched and merged to form the final shape.

While through sketching bending strokes with the new approach we can model a large variety of freeform shapes with desired curvature patterns, there obviously remain many more shapes that cannot be easily created with this approach alone. Our technique makes a valuable complement to the diverse and powerful toolbox of geometric modeling through sketching and interactions.

**Limitations.** To model each surface patch, we use a height-field representation which cannot handle self-occluding shapes or layered objects. To mitigate this limitation, we provide the multi-view framework that allows sketching patches from different viewpoints and merging them to form more complex shapes. Still, the incrementally added new patches have to be anchored, by either referencing the previous patches, or relying on the user specification of a spatial wireframe, which is nontrivial and targeted by numerous works [Bae et al. 2008; Schmidt et al. 2009; Xu et al. 2014]. Thus we emphasize that our work complements and is best integrated with other sketch-based modeling methods.

**Future work.** The current approach and its user interface can be improved in many aspects. For example, with pressure sensitive pens the sketching of bending strokes can naturally incorporate the curvature scaling ratio. The images if available can be used to extract information like boundary or shading to assist the shape creation process. The texturing and animation of the created 3D shapes are highly desirable features for a more complete modeling tool. We would also like to explore how to better integrate the proposed method with existing 3D modeling techniques and systems.

## APPENDIX

*Discretization of  $dN$ .* Because the shape operator  $dN = -\text{II}G^{-1}$ , we can discretize it through defining the first and second fundamental forms of the height surface  $z = f(x, y)$  discretized as a piecewise linear function over the triangle mesh domain  $M$ .

For each triangle  $\Delta_{ijk} \in M$  with vertices indexed by  $i, j, k$ , the gradient of height field function over the triangle is ([Botsch et al. 2010])

$$\nabla f = (f'_x, f'_y)^T = (f_j - f_i) \frac{(\mathbf{x}_i - \mathbf{x}_k)^\perp}{2A_{ijk}} + (f_k - f_j) \frac{(\mathbf{x}_j - \mathbf{x}_i)^\perp}{2A_{ijk}}. \quad (6)$$

The unit normal vector for the lifted triangle is thus  $\mathbf{n} = (n_1, n_2, n_3)^T = \frac{(f'_x, f'_y, -1)}{\sqrt{f'^2_x + f'^2_y + 1}}$ .

The first fundamental form is readily computed by

$$G = \begin{pmatrix} 1 + (\frac{\partial f}{\partial x})^2 & \frac{\partial f}{\partial x} \cdot \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial x} \cdot \frac{\partial f}{\partial y} & 1 + (\frac{\partial f}{\partial y})^2 \end{pmatrix}. \quad (7)$$

The second fundamental form  $\text{II} = n_3 H$  involves the Hessian  $H$  which has to be defined for a triangle mesh. Since Hessian is simply the Jacobian of gradient, we have  $\delta \mathbf{x}^T H \delta \mathbf{x} = \delta \mathbf{x}^T \delta \nabla f$ , where  $\delta \mathbf{x}$  denotes a 2D tangential move, and  $\delta \nabla f$  the corresponding change of gradient. For the discrete setting (see Fig. 14),  $\delta \mathbf{x}$  denotes the vector from center of one triangle to that of its neighbor, and  $\delta \nabla f$  the difference of the gradients at the two triangles.

Note that since  $\delta \mathbf{x}$  is determined by the triangulation, and  $\delta \nabla f$  is a linear function of vertex heights, the equation for Hessian is

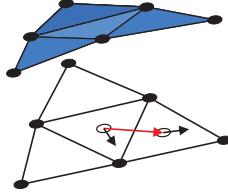


Fig. 14. The tangential movement (red vector) and the corresponding change of vector field (the two black vectors located at the two triangles). The tangential movement starts from the circumcenter of the current triangle, and points to the circumcenter of the neighbor triangle. The vector field can be the gradient field, or the principal curvature direction field, and is defined constant per triangle of the domain mesh.

indeed also linear with respect to Hessian entries and the vertex heights. In particular, suppose  $H = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$ ,  $\delta x = (\delta_x, \delta_y)^T$ , and  $\delta \nabla f = (g_x, g_y)^T$ , we have:

$$\begin{pmatrix} \delta_x^2, 2\delta_x\delta_y, \delta_y^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \delta_x g_x + \delta_y g_y. \quad (8)$$

For each inner face with three neighbor triangles sharing common edges, we have three such equations which are assembled into a linear equation system  $Ay = b$ , with  $y = (a, b, c)^T$ ; the linear system is solved as  $y = A^{-1}b$ , so we get  $H$  again as a linear function of vertex heights appearing in the vector  $b$ .

To minimize the nonlinear functional  $E_{match}$ , in one iteration we assume the normal vectors are fixed; thus  $G$  is known, and  $\Pi = n_3 H$  is a linear function of vertex heights, making  $dN$  a linear function as well. As a result,  $E_{match}$  is linearized in the iteration, and can be solved efficiently.

*Discretization of  $\nabla_u v$ .* In our planar domain setting,  $\nabla_u v = \nabla v \cdot u$ , where  $\nabla v = \begin{pmatrix} \nabla v_x^T \\ \nabla v_y^T \end{pmatrix}$  consists of the gradients of the two components of  $v$ . Next we derive  $\nabla v$  on the triangle mesh as a linear function of  $v$ .

By definition, we have  $\nabla v \delta x = \delta v$ , where  $\delta x$  denotes a 2D tangential move, and  $\delta v$  the corresponding change of  $v$ . For the discrete setting (Fig. 14),  $\delta x$  again is the vector from the center of one triangle to that of its neighbor, and  $\delta v$  the difference of  $v$  on the two triangles.

For a triangle with two or three neighbors sharing common edges, each pair formed by the triangle and one of its neighbors leads to an equation, thus we have  $\nabla v \delta x_j = \delta v_j, j = 1, 2$  or  $1, 2, 3$ . We assemble the equations into a system  $\nabla v A = B$ , with  $A = (\delta x_1, \delta x_2, \dots)$  and  $B = (\delta v_1, \delta v_2, \dots)$ . Then we solve for  $\nabla v$  in the least square sense  $\nabla v = BA^+$ , where  $A^+$  is the right pseudo-inverse matrix of  $A$ . Since  $\delta x$  (and thus  $A$ ) is known,  $\nabla v$  is indeed solved as a linear function of  $v$  appearing in  $B$ .

*Harmonic non-orthogonal 4-direction field computation.* In the PolyVector representation [Diamanti et al. 2014], the pair of directions  $u, v$  of a non-orthogonal 4-direction field are converted to unit complex numbers  $u, v \in \mathbb{C}$ , and the direction field is encoded by another pair of complex variables  $a = u^2 + v^2$  and  $b = u^2 v^2$ , which is derived from the coefficients of the polynomial equation  $(z^2 - u^2)(z^2 - v^2) = 0, z \in \mathbb{C}$ .

Given the strokes, the harmonic 4-direction field is found by solving the following optimization problem:

$$\text{Minimize } E_{field}(a, b) = E_{smooth} + \omega_c E_{constraint} + \omega_o E_{ortho} \quad (9)$$

with  $\omega_c > 0$  the weight of directional constraint, and  $\omega_o$  the weight of orthogonality of the  $u, v$  directions. We used  $\omega_c = 10^3$ ,  $\omega_o = 10^{-5}$  in our implementation. The direction field smoothness energy is

$$E_{smooth} = \frac{1}{|\Omega|} \sum_{f \sim g} |e_{fg}| \cdot (|a_f - a_g|^2 + |b_f - b_g|^2), \quad (10)$$

where the pair of triangles  $f$  and  $g$  are adjacent at the edge  $e_{fg}$ , and  $|e_{fg}|$  denotes the area of triangles formed by the edge and the two centroids of  $f, g$ .

The stroke direction constraint on the direction field is defined by plugging the stroke direction into the polynomial equation and taking the residual:

$$E_{constraint} = \frac{1}{|\Omega|} \sum_f |f| \cdot |u_f^2 a_f - b_f - u_f^4|^2, \quad (11)$$

where  $f$  indexes the triangles with stroke constraints, and  $u_f \in \mathbb{C}$  is the tangent direction of the stroke crossing the triangle.

The very mild requirement of orthogonality for the field directions is here to make sure the directions can be solved even when the constraints are biased and not sufficient for determining the two sets of crossing directions, for example, when the input strokes are a set of parallel straight lines. The energy is simply a penalization of the magnitude of  $a$ :

$$E_{ortho} = \frac{1}{|\Omega|} \sum_{f \in M} |f| \cdot |a_f|^2. \quad (12)$$

Note the optimization is a standard quadratic optimization, which can be solved efficiently through solving a linear equation system. After the optimal solution  $a, b$  is found for the harmonic direction field, the  $u, v$  directions are recovered from them.

## ACKNOWLEDGMENTS

The authors would like to thank the authors of [Iarussi et al. 2015] for providing source code for comparison, the user evaluation participants from Microsoft Research Asia, Tsinghua University and Beijing Film Academy, Rui Xia and Doris Hoogeveen for preparing video, and the reviewers for their suggestions.

The work of Wenping Wang was partially supported by the National Basic Research Program of China (2011CB302400), NSFC (61272019), the Research Grant Council of Hong Kong (717012, 1720915, 17211017) and sponsorship from Microsoft (AR160005). The work of Alla Sheffer was supported by NSERC.

## REFERENCES

- 2016. CGAL, Computational Geometry Algorithms Library. <http://www.cgal.org>. (2016).
- 2016. ZBrush. <http://pixologic.com/>. (2016).
- Nina Amenta and Marshall Bern. 1998. Surface Reconstruction by Voronoi Filtering. In *SoCG*, 39–48.
- Alexis Andre and Suguru Saito. 2011. Single-view Sketch Based Modeling. In *SBIM*, 133–140.
- Alexis Andre, Suguru Saito, and Masayuki Nakajima. 2007. CrossSketch: Freeform Surface Modeling with Details. In *SBIM*, 45–52.
- Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. 2008. ILoveSketch: As-natural-as-possible Sketching System for Creating 3D Curve Models. In *UIST*, 151–160.
- Jonathan T Barron and Jitendra Malik. 2015. Shape, Illumination, and Reflectance from Shading. *IEEE Trans. Pattern Anal. Mach. Intell.* 37 (2015), 1670–1687.

- Peter N. Belhumeur, David J. Kriegman, and Alan L. Yuille. 1999. The Bas-Relief Ambiguity. *Int. J. Comput. Vis.* 35, 1 (1999), 33–44.
- Adrien Bernhardt, Adeline Pihuit, Marie-Paule Cani, and Loïc Barthe. 2008. Matisse : Painting 2D regions for Modeling Free-Form Shapes. In *SBIM*. 57–64.
- Mikhail Bessmeltsev, Will Chang, Nicholas Vining, Alla Sheffer, and Karan Singh. 2015. Modeling Character Canvases from Cartoon Drawings. *ACM Trans. Graph.* 34, 5 (2015).
- Mikhail Bessmeltsev, Caoyu Wang, Alla Sheffer, and Karan Singh. 2012. Design-driven Quadrangulation of Closed 3D Curves. *ACM Trans. Graph. (SIGGRAPH ASIA)* 31, 6 (2012), 178:1–178:11.
- M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Levy. 2010. *Polygon Mesh Processing*. Taylor & Francis.
- Minh Tuan Bui, Junho Kim, and Yunjin Lee. 2015. 3D-look Shading from Contours and Hatching Strokes. *Comput. Graph.* 51 (2015), 167–176.
- Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 2013. 3-Sweep: Extracting Editable Objects from a Single Photo. *ACM Trans. Graph. (SIGGRAPH ASIA)* 32, 6 (2013), 195:1–195:10.
- Pedro Company, Ana Piquer, Manuel Contero, and Ferran Naya. 2005. A Survey on Geometrical Reconstruction As a Core Technology to Sketch-based Modeling. *Comput. Graph.* 29, 6 (2005), 892–904.
- Frederic Cordier, Hyewon Seo, Jinho Park, and Jun Yong Noh. 2011. Sketching of Mirror-Symmetric Shapes. *IEEE T. Vis. Comput. Gr.* 17, 11 (2011).
- Chris De Paoli and Karan Singh. 2015. SecondSkin: Sketch-based Construction of Layered 3D Models. *ACM Trans. Graph. (SIGGRAPH)* 34, 4, Article 126 (2015), 10 pages.
- Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. 2014. Designing N-PolyVector Fields with Complex Polynomials. *Comput. Graph. Forum (SGP)* 33, 5 (2014), 1–11.
- Chao Ding and Ligang Liu. 2016. A survey of sketch based modeling systems. *Frontiers of Computer Science* 10, 6 (2016), 985–999.
- Even Entem, Loïc Barthe, Marie-Paule Cani, Frederic Cordier, and Michiel Van De Panne. 2014. Modeling 3D animals from a side-view sketch. *Comput. Graph.* 38 (2014).
- Hongbo Fu, Yichen Wei, Chiew-Lan Tai, and Long Quan. 2007. Sketching Hairstyles. In *SBIM*. 31–36.
- Yotam Gingold, Takeo Igarashi, and Denis Zorin. 2009. Structured Annotations for 2D-to-3D Modeling. *ACM Trans. Graph. (SIGGRAPH ASIA)* 28, 5 (2009), 148:1–148:9.
- Yotam Gingold and Denis Zorin. 2008. Shading-based Surface Editing. *ACM Trans. Graph. (SIGGRAPH)* 27, 3 (2008), 95:1–95:9.
- Aaron Hertzmann and Denis Zorin. 2000. Illustrating Smooth Surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. 517–526.
- Haibin Huang, Evangelos Kalogerakis, Ersin Yumer, and Radomir Mech. 2016. Shape Synthesis from Sketches via Procedural Models and Convolutional Networks. *IEEE T. Vis. Comput. Gr.* 22, 10 (2016), 1.
- Emmanuel Iarussi, David Bommes, and Adrien Bousseau. 2015. BendFields: Regularized Curvature Fields from Rough Concept Sketches. *ACM Trans. Graph. (SIGGRAPH)* 34, 3 (2015), 24:1–24:16.
- Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. 1999. Teddy: A Sketching Interface for 3D Freeform Design. In *SIGGRAPH*.
- Joaquim Jorge and Faramarz Samavati (Eds.). 2011. *Sketch-based Interfaces and Modeling* (1 ed.). Springer-Verlag London.
- Pushkar Joshi and Nathan A. Carr. 2008. Repoussé: Automatic Inflation of 2D Artwork. In *SBIM (SBM'08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 49–55.
- P. Joshi and C. H. Séquin. 2007. Energy Minimizers for Curvature-Based Surface Functionals. *CAD Conference* (2007), 607–617.
- Amaury Jung, Stefanie Hahmann, Damien Rohmer, Antoine Begault, Laurence Boissieux, and Marie-Paule Cani. 2015. Sketching Folds: Developable Surfaces from Non-Planar Silhouettes. *ACM Trans. Graph.* 34, 5, Article 155 (2015), 12 pages.
- Olga A. Karpenko and John F. Hughes. 2006. SmoothSketch: 3D Free-form Shapes from Complex Sketches. *ACM Trans. Graph. (SIGGRAPH)* 25, 3 (2006), 589–598.
- Jan K. Koenderink and Andrea J. van Doorn. 2002. Image Processing Done Right. In *ECCV*. 158–172.
- Vladimir Kolmogorov. 2006. Convergent Tree-Reweighted Message Passing for Energy Minimization. *IEEE Trans. Pattern Anal. Mach. Intell.* 28, 10 (2006), 1568–1583.
- Z. Li, J. Liu, and X. Tang. 2007. A Closed-form Solution to 3D Reconstruction of Piecewise Planar Objects from Single Images. In *CVPR*. 1–6.
- Yang Liu, Hao Pan, John Snyder, Wenping Wang, and Baining Guo. 2013. Computing Self-supporting Surfaces by Regular Triangulation. *ACM Trans. Graph.* 32, 4 (2013), 92:1–92:10.
- Yongwei Miao, Feixia Hu, Xudong Zhang, Jiazhou Chen, and Renato Pajarola. 2015. SymmSketch: Creating symmetric 3D free-form shapes from 2D sketches. *Computational Visual Media* 1, 1 (2015), 3–16.
- Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. 2007. FiberMesh: Designing Freeform Surfaces with 3D Curves. *ACM Trans. Graph. (SIGGRAPH)* 26, 3 (2007).
- Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. 2005. A Sketch-based Interface for Detail-preserving Mesh Editing. *ACM Trans. Graph. (SIGGRAPH)* 24, 3 (2005), 1142–1147.
- Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. 2016. Interactive Sketching of Urban Procedural Models. *ACM Trans. Graph. (SIGGRAPH)* 35, 4 (2016), 130:1–130:11.
- L. Olsen, F. Samavati, and J. Jorge. 2011. NaturaSketch: Modeling from Images and Natural Sketches. *IEEE Comput. Graph. Appl. Mag.* 31, 6 (2011), 24–34.
- Luke Olsen, Faramarz F. Samavati, Mario Costa Sousa, and Joaquim A. Jorge. 2009. Sketch-based modeling: A survey. *Comput. Graph.* 33, 1 (2009), 85 – 103.
- Helmut Pottmann and Yang Liu. 2007. Discrete Surfaces in Isotropic Geometry. In *Mathematics of Surfaces XII*. 341–363.
- Cody Robson, Ron Mahrik, Alla Sheffer, and Nathan Carr. 2011. Context-aware Garment Modeling from Sketches. *Comput. Graph.* 35, 3 (2011), 604–613.
- Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. 2009. Analytic Drawing of 3D Scaffold. *ACM Trans. Graph. (SIGGRAPH ASIA)* 28, 5 (2009), 149:1–149:10.
- Ryan Schmidt, Brian Wyvill, Mario Costa Sousa, and Joaquim A. Jorge. 2005. ShapeShop: Sketch-Based Solid Modeling with BlobTrees. In *SBIM*.
- Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. 2012. CrossShade: Shading Concept Sketches Using Cross-section Curves. *ACM Trans. Graph. (SIGGRAPH)* 31, 4 (2012), 45:1–45:11.
- Alex Shtof, Alexander Agathos, Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. 2013. Geosemantic Snapping for Sketch-Based Modeling. *Comput. Graph. Forum (EG)* 32, 2 (2013), 245–253.
- Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. 2004. Laplacian Surface Editing. In *Symp. Geom. Proc.* 179–188.
- Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Brian Whited, Maryann Simmons, and Olga Sorkine-Hornung. 2014. Ink-and-Ray: Bas-Relief Meshes for Adding Global Illumination Effects to Hand-Drawn Characters. *ACM Trans. Graph.* 33, 2 (2014), 16.
- Chiew-Lan Tai, Hongxin Zhang, and Jacky Chun-Kin Fong. 2004. Prototype Modeling from Sketched Silhouettes based on Convolution Surfaces. *Comput. Graph. Forum* 23, 1 (2004), 71–83.
- Emmanuel Turquin, Marie-Paule Cani, and John F. Hughes. 2004. Sketching Garments for Virtual Characters. In *SBIM*. 175–182.
- Etienne Vouga, Mathias Höbinger, Johannes Wallner, and Helmut Pottmann. 2012. Design of Self-supporting Surfaces. *ACM Trans. Graph. (SIGGRAPH)* 31, 4 (2012), 87:1–87:11.
- Yingze Wang, Y. Chen, J. Liu, and X. Tang. 2009. 3D reconstruction of curved objects from single 2D line drawings. In *CVPR*. 1834–1841.
- Xiaohua Xie, Kai Xu, Niloy J. Mitra, Daniel Cohen-Or, Wenyong Gong, Qi Su, and Baoquan Chen. 2013. Sketch-to-Design: Context-Based Part Assembly. *Comput. Graph. Forum* 32, 8 (2013), 233–245.
- Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. 2014. True2Form: 3D Curve Networks from 2D Sketches via Selective Regularization. *ACM Trans. Graph. (SIGGRAPH)* 33, 4 (2014), 131:1–131:13.
- Kun Xu, Kang Chen, Hongbo Fu, Wei-Lun Sun, and Shi-Min Hu. 2013. Sketch2Scene: Sketch-based Co-retrieval and Co-placement of 3D Models. *ACM Trans. Graph. (SIGGRAPH)* 32, 4 (2013), 123:1–123:15.
- Q. Xu, Y. Gingold, and K. Singh. 2015. Inverse Toon Shading: Interactive Normal Field Modeling with Isophotes. In *SBIM*. 15–25.
- C. K. Yeh, S. Y. Huang, P. K. Jayaraman, C. W. Fu, and T. Y. Lee. 2016. Interactive High-Relief Reconstruction for Organic and Double-sided Objects from a Photo. *IEEE T. Vis. Comput. Gr.* 99 (2016), 1–1.
- Li Zhang, G. Dugas-Phocion, J. S. Samson, and S. M. Seitz. 2001. Single view modeling of free-form scenes. In *CVPR*, Vol. 1. I–990–I–997.
- L. Zhu, T. Igarashi, and J. Mitani. 2013. Soft Folding. *Comput. Graph. Forum (PG)* 32, 7 (2013), 167–176.
- Yixin Zhuang, Ming Zou, Nathan Carr, and Tao Ju. 2013. A General and Efficient Method for Finding Cycles in 3D Curve Networks. *ACM Trans. Graph. (SIGGRAPH ASIA)* 32, 6 (2013), 180:1–180:10.