SpineLoft: Interactive Spine-based 2D-to-3D Modeling

Alexandre Thiault, Telo Philippe, Amal Dev Parakkat, Elmar Eisemann, Ramanathan Muthuganapathy, Takeo Igarashi

SpineLoft: Interactive Spine-based 2D-to-3D Modeling

Alexandre Thiault
LTCI-Telecom Paris, IP Paris
Palaiseau, France
alexandre.thiault@hotmail.com

Telo Philippe
LTCI-Telecom Paris, IP Paris
Palaiseau, France
telo.philippe@gmail.com

Amal Dev Parakkat
LTCI-Telecom Paris, IP Paris
Palaiseau, France
adp.upasana@gmail.com

Elmar Eisemann
Delft University of Technology
Delft, Netherlands
e.eisemann@tudelft.nl

Ramanathan Muthuganapathy
Indian Institute of Technology Madras
Chennai, India
emry01@gmail.com

Takeo Igarashi
The University of Tokyo
Tokyo, Japan
takeo@acm.org

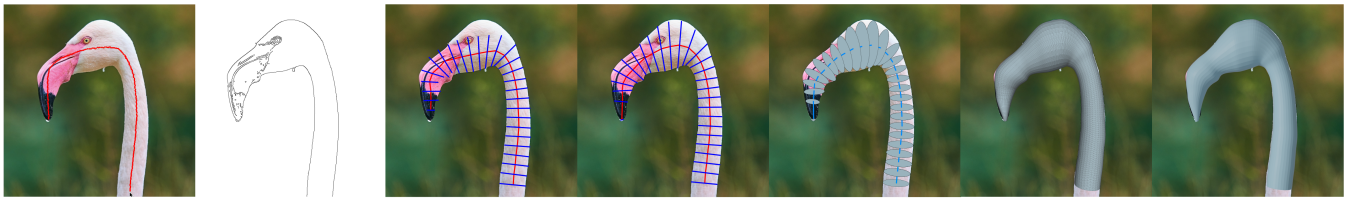


Figure 1: Our system takes an image along with user annotations to compute an editable spine-rib system. Based on detected edges and a user-defined spine, it generates a 3D model, extruding a user-defined cross-section (here, the cross-section was chosen to be circular). Our solution addresses inherent problems of image-based systems (like missing edges and occluded regions), and users can modify the geometry locally (Image from PixaBay - www.pixabay.com).

Abstract

3D artists (professionals and novices alike) often take inspiration from sketches or photos to guide their designs. Yet, existing modeling systems are not tailored to fully make use of such input. Consequently, significant effort and expertise are needed when creating model prototypes or exploring design options. In this work, we introduce a system to support the exploratory modeling process by enabling the transformation of 2D image elements into geometric 3D objects. Our solution relies on a novel d_2 distance function, supporting a region-based lofting process, and delivers easily-editable 3D geometric "spine-rib" representations. The user draws a spine, and the system generates and modifies a generalized cylinder around it, considering image edges. The proposed approach, driven by simple user-defined scribble definitions, can robustly handle various image sources, ranging from photos to hand-drawn content.

CCS Concepts

• **Applied computing** → **Arts and humanities**; • **Theory of computation** → **Computational geometry**; • **Computing methodologies** → **Shape modeling**; • **Human-centered computing** → **Interactive systems and tools**.

Keywords

Sketch-based 3D modeling, Image-based 3D modeling, d_2 function, Lofting, Interactive modeling

ACM Reference Format:
Alexandre Thiault, Telo Philippe, Amal Dev Parakkat, Elmar Eisemann, Ramanathan Muthuganapathy, and Takeo Igarashi. 2025. SpineLoft: Interactive Spine-based 2D-to-3D Modeling. In . ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3706598.3713439

1 Introduction

Sketch-based modeling has gained much attention since it is typically easier to sketch in 2D than directly working on a 3D object. A sketch can guide an artist during modeling, and it is even common to start with a 2D concept sketch, often involving existing image sources for inspiration. Still, there is a separation between the 2D information and the actual 3D modeling step. Our approach, SpineLoft, will bring these two domains closer together by allowing artists (professionals or novices) to transform 2D regions, even if coarsely defined in a sketch or partially occluded, into a 3D element to be used in their model design, relying only on simple user annotations. To make our solution effective, we address the following questions:

- Selection: How to easily support selecting regions of interest from image references?
- Robustness: How to handle adverse conditions (occluded, missing or ambiguous boundaries)?
- Editing: How to provide the possibility to influence the creation of 3D geometry in an intuitive manner?

To address region selection, we present a novel distance function to create a hull. The user provides a scribble (*spine*), for which we generate a set of outgoing edges around it (*ribs*), which, together with user-defined cross sections, results in a 3D representation, conceptually similar to an endoskeleton. All annotations can be loose

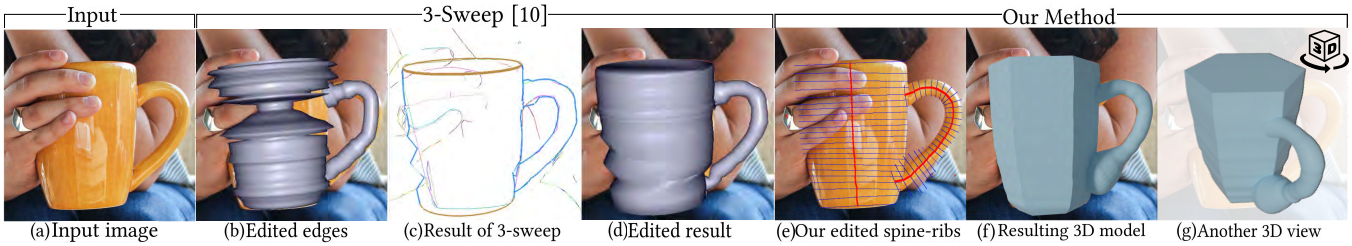


Figure 2: Comparison with the state-of-the-art method (3-sweep [10]). 3-sweep tends to fail on shapes with ambiguous edges, such as this mug with a jagged shape profile (b), requiring the user to edit the input image extensively (c). In contrast, our method based on a spine-rib representation (e) is more robust and generates a more plausible and smoother shape (f, g)

as SpineLoft automatically optimizes them following the image content. To achieve robustness, we rely on a rib length optimization to handle partial occlusion or noisy boundaries, which are especially common in hand-drawn content and photographs. Specifically, our procedure follows an optimization that targets a smooth variation of the ribs while trying to respect the region boundaries from the image.

To address editing, we do allow user interaction to change the rib length (either individually or as a group). Similarly, SpineLoft supports user-defined cross sections (either drawn or selected from a predefined set), which help influence the volumetric aspect of an object inspired by traditional "lofting" techniques. Finally, we enable spine deformations to ease the composition of different elements; an existing spine-rib representation can be copied to a new spine. This action can also be used to drive animations.

The main objective of SpineLoft is to aid novice users who are new to 3D modeling. Traditional 3D modeling systems based on polygonal modeling (i.e., as used in the popular software Blender) require users to interact with a 3D scene and to understand the underlying 3D shape representation (polygons), which is challenging for novices. In SpineLoft, we aim to alleviate these difficulties by enabling users to create 3D models with simple 2D interactions that are quick to perform. From these simple 2D interactions, we create a spine-rib system that helps bootstrap the creation process yet retains editability over the final 3D shape. While the focus of SpineLoft is to encourage creativity and exploration among novice users, it can also be used by advanced users. It can serve for rapid prototyping before refining the results further in advanced modeling software. In summary, our work makes the following contributions:

- A novel explicit d_2 distance function to compute non-intersecting gradient lines from a user-drawn spine. This explicit computation is both easy to implement and efficient, making it readily reusable for various interactive tasks, offering advantages over the widely used Euclidean distance function.
- A region extraction algorithm relying on user annotations, which can address noisy or missing edges in the input, making it useful in creating interactive image cut-out tools similar to Lazy Snapping [35].
- A related lofting method built on a rib length optimization to quickly create 3D shapes from erroneous images.
- An interface to define/interact with the spine-rib representation (a novel representation for 3D modeling), handling discrepancies, like occlusion/noise/missing data, with respect to the reference image. Additionally, it prioritizes editability, recognizing that novice users are more likely to make errors.

2 Related Works

The related work for SpineLoft can be classified into two categories: Sketch-based 3D modeling and Playful Interfaces.

Sketch-based 3D modeling: Sketch-based 3D modeling literature is too vast to cover completely in this paper, which is why we restrict ourselves to the most-related solutions and refer the interested reader to various existing surveys [5, 6, 31, 41, 62].

Teddy [23] is a seminal system where a fixed input boundary is inflated to create a 3D shape. The method can be extended to support general input images [8], relighting [43] and animation [7]. The latter papers, RigMesh [7] and MonsterMash [16], generate 3D models by assembling parts created in a single-view modeling interface. While these two approaches rely on circular cross-sections, NaturaSketch [40] proposes a simple inflation mechanism that involves a user-defined distance function to modify the object's cross-section. Andre et al.[1] use a user-drawn boundary stroke and scaling factor to define a sweeping surface. Yet, the input has to be drawn from a fixed viewpoint - making it difficult for novice users. Peng et al. [45] introduced a sculpting-based system with a focus on animation, but it is mainly useful for repetitive spatiotemporal tasks.

CreatureShop [61] allows users to define regions in an input image but uses simple inflation. Bernhardt et al. [2] use painted 2D regions in an implicit-based 3D modeling approach, giving control over the blending, depth, and thickness.

Gingold et al. [19] used a generalized cylinder fitting based on user annotations to create the desired model. Shtof et al. [51] introduced an interactive geometric snapping tool relying on a simple drag-and-drop modeling interface. 3-Sweep [10] extends the method to extract and manipulate objects in a single photograph. While being an inspiration, 3-Sweep is limited with respect to edits and control over cross-sections. Further, the effect of occlusions or missing edges can lead to undesired artifacts (Figure 2). It is worth noting that these methods often rely on a Euclidean distance function, which, as explained in Section 3.1, might not always give the desired results.

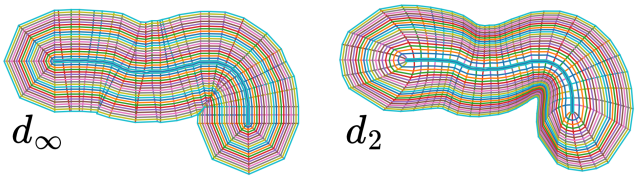


Figure 3: Comparison of the d_∞ and d_2 gradient lines associated with a spine represented in dark blue. Isolines show equal distance values. d_2 results in more smoothness and avoids merges inside concavities. Whereas, the d_∞ function leads to singular gradients at some points in space and thus would yield intersecting ribs.

Some solutions are less general, requiring 3D skeletons [3], or focusing on particular content, like garments [18, 48], or trees [11], animals in a side view [17], or animal heads [37]. Other similar works that are worth mentioning include the use of 3D scaffolds [25, 26] and reference RGB-D images [34] to create 3D models, but are typically targeting expert users, take much time and effort, or are designed only for initial prototyping.

Deep learning has had a major impact on 2D-to-3D modeling tasks. Including sketch-based retrieval [59], single-view automatic 3D modeling [21], single-view interactive 3D modeling [32], normal estimation techniques [22] and multi-view modeling [15]. However, user control and related editing are limited for these cases.

Mesh deformation is a well-studied topic in 2D [9] and 3D [24], including advanced deformation techniques using multistroke contour drawings [28] or pose/gesture drawings [4, 20]. Nevertheless, few of these techniques are integrated directly into the creation process, which is crucial for prototypical modeling as targeted in this work.

Different from traditional interactive modeling techniques [16, 19, 40, 61], which take images as a reference over which the user has to trace the desired shape (a time-consuming task), our objective is to utilise cues extracted from the input photograph (from the wild) to ease the modeling process. Further, it is worth mentioning that though sketches act as an intuitive and simple medium for 3D modeling, it is not restricted to these alone. Many systems combine user inputs with computer-vision techniques to create 3D models from various sources, such as multi-view stereo [46], multi-view images [60], unordered photo collections [53] and videos [57]. Another important direction involves using geometric constraints [36] or interactive sculpting [14, 54] to iteratively refine a basic shape into the desired 3D model.

Playful Interfaces: Thanks to the tools that enhance user engagement and enjoyment in a playful exploratory manner [47], the concept of "Playful Interfaces" has gained attention in HCI research. Not only are such interfaces accessible to novice users (including children),but they also improve user experience by providing an appealing and intuitive interaction. While the literature has explored such interfaces for a variety of tasks, such as creative design [30], sketch processing [42], color interaction [52], and programming [38], their application in the context of sketch-based modeling for novice users remains a promising direction. Such playful interfaces can lower the entry barrier for 3D modeling, making it more

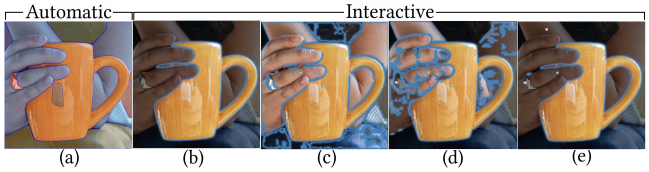


Figure 4: Result of Segment Anything Model (SAM) [27] on the image shown in Figure 2. (a) The automatic segmentation and (b-e) different steps of interactive segmentation.

enjoyable and less intimidating while potentially increasing user motivation, encouraging experimentation and, ultimately, leading to improved learning outcomes. In contrast to works in this direction [23], our main objective is to further simplify the modeling process by providing users with the support to draw inspiration from existing images/photographs. It is worth noting that these images/photographs serve only as references while giving complete creative freedom to the user. They are supported in conceptualizing their ideas while being encouraged to explore and experiment (please refer to Section 4).

3 Spine-rib based modeling

3.1 Design Rationale

Inspired by skeletal systems widely seen in many organic shapes, we adopt such a structure for our intuitive approach to sketch-based 3D modeling. Our spine-rib system allows users to easily conceptualize 3D shapes by focusing on a central axis (spine) and its associated cross-sections (ribs). The simplicity of this representation makes it accessible to novice users with little to no modeling experience, enabling them to create 3D models with minimal input. The modeling process starts with the user drawing an approximate spine of the object to be modeled (an easy task to do, thanks to the flexibility to draw imprecise spines and the natural ability of users to infer spines). Once the spine is drawn, the system can then compute the corresponding ribs - automating a significant portion of the modeling process.

Though ribs can be imagined as line segments orthogonal to the spines, automatically computing them is not trivial. The simple solution for computing ribs would be to follow the gradient of a simple Euclidean distance function, which, while intuitive, is not differentiable everywhere (because the function min is not). Its gradient discontinuities correspond to the local maxima of the distance function. Consequently, multiple points that follow the gradient from different starting positions can converge to the same discontinuity, causing intersections (as shown in Figure 3). Yet, more complex distance functions based on heat equations, while avoiding intersections, can be computationally expensive.

To address these challenges, we introduce a d_2 function that is differentiable and has continuous gradients to provide a smoother and more stable gradient field. Using our d_2 distance function w.r.t. the user-drawn spine, we can define ribs as the two points ascending the gradient that will follow parallel paths when approaching each other, preventing intersections. This function allows for the automatic computing of non-intersecting ribs, making the modeling process more accessible and less error-prone for novice users.

During modeling, these ribs can then act as a guide for a lofting surface. As the ribs are just lines projecting out of the spine, we determine endpoints based on their intersection with reference image edges. However, due to occlusions or variations in image intensity, accurately identifying correct rib endpoints can be challenging. Even powerful segmentation tools like SAM (Segment Anything Model) [27] may struggle to consistently and precisely identify the required boundaries, as shown in Figure 4.

To overcome these challenges posed by the inconsistencies in the input reference image, we implement a rib length optimization technique to eliminate noisy or erratic ribs. As a consequence, we can reduce the need for manual corrections and create a cleaner and more coherent 3D model. Nevertheless, we also provide an interactive rib editing functionality where the users can click and drag individual ribs or edit multiple ribs simultaneously to provide flexibility. With this balance of an automatic approach and interactive editing, we ensure that the final 3D model aligns with the user's intentions.

3.2 Overview

The overview of SpineLoft is illustrated in Figure 1. The user selects an input image, which can be photos, illustrations, or sketches. Then, a region of interest, which is to be converted, is selected by having the user draw a scribble (referred to as *spine* - in the spirit of curvy skeletons [3]) along the region. From the spine, outgoing edges (referred to as *ribs*) are generated that respect the boundary of the region but can be user-adjusted. From this input, the method follows a lofting procedure to derive a corresponding 3D shape of the modeled part.

SpineLoft has been built with ease of use in mind. Therefore, we need to robustly process the image input, handling missing edges or noise. Further, imperfect user input will be common and should still lead to a successful lofting process, which requires the spine to be adapted and the generated ribs to be constructed carefully.

In the following, we will describe the steps of our solution in detail. We first explain how to produce ribs in an iterative process. We take steps from the spine along a suitable path (Sec. 3.3.1) until reaching a region boundary, as indicated by an edge detector. To handle occlusions and imperfections in the input image, we rely on a rib length optimization procedure (Sec. 3.3.2). To allow for larger expressiveness, the user can also interact with the resulting rib-spine system (Sec. 4). Finally, the original spine is improved based on the computed extent of the ribs, and a final 3D shape is generated. The latter is obtained by weaving a cross-section along the spine, following its orientation and using the ribs to determine the scale (Sec. 3.4).

3.3 Technical details

3.3.1 ***Generating Ribs.*** The rib construction starts with the user drawing an initial spine on top of the reference image (without self-intersections or loops, and approximately going through the center of the required region). The spine consists of points that are defined by 2D-pixel coordinates along the curve. Yet, it would be insufficient to simply extend the ribs orthogonally outward from these spine points, as it could lead to intersections that will not result in a valid lofted geometry.

Instead, we offset these k spine points only by a value of ϵ in both normal directions (for our experiments, ϵ is set to half the minimum distance between two consecutive spine points), which allows us to construct a hull H_0 composed of $2 * k$ points around the spine (the blue polygon in the center of Figure 3). To avoid rib crossings, we will define a distance function to H_0 in image space. The gradient of this distance function will be used to drive the rib generation (where each 'rib' is associated with a single distance value). Starting from the hull H_0, we iteratively follow the gradient, using an Euler method with an adaptive step size depending on the gradient's magnitude. This trajectory will define *gradient curve*. Naturally, following the gradient will avoid rib intersections and make them initially orthogonal to H_0's boundary.

Unfortunately, using a standard distance function between a point x and surface M [44], defined as:

$$d(x, M) = \inf_{y \in M} |x - y|$$

where y represents points on M, does not provide an explicit solution in 2D. Related alternatives [58] typically result in coarse distance approximations, which leads to a significant loss of small-scale spine features. Instead, we define a natural generalized distance function between a point x and a polygon P, consisting of vertices P_0 to P_{k-1}, with perimeter $A = \sum_{i=0}^{k-1} \|P_{i+1} - P_i\|$.

The distance function of degree n between x and P is then defined as an integral on the contour of P [44]:

$$d_n(x, P) = A^{1/n} \left(\int_P \|x - y\|^{-n} \, dy \right)^{-1/n}$$

which when $n = 2$, evaluates to:

$$d_2(x, P) = \sqrt{A} \left(\int_P \|x - y\|^{-2} \, dy \right)^{-1/2}$$

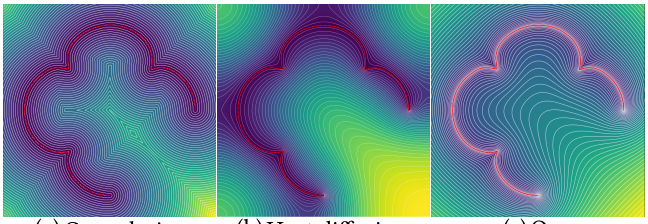
Different from Peng et al. [44], which used a d_3 function (in a 3D configuration), we use $n = 2$, as it results in an explicit formulation while yielding good results and being efficient/easy to use.

In this section, we explain the discrete formulation of our d_2 function, and we redirect the reader to Appendix A for the complete derivation. When the user draws a spine, we consider it a polyline with an ordered set of points p_i, with i ranging from 0 to n. We aim to compute the d_2 distance between a point x and the curve (user-drawn spine). Due to the discrete nature of the curve, we use a discrete sum:

$$d_2(x) = \frac{\sqrt{A}}{\sqrt{\int_{\text{Curve}} \|x - y\|^{-2} \, dy.}} = \frac{\sqrt{A}}{\sqrt{\sum_{i=0}^{n-1} Int_{[p_i, p_{i+1}]}}}$$

where for each segment:

$$\begin{aligned} Int_{[p_i, p_{i+1}]} &= \int_{p_i}^{p_{i+1}} \|x - y\|^{-2} \, dy \\ &= \int_0^T \|q_0 - t q_1\|^{-2} \, dt \quad \text{after integrating by substitution} \\ &\quad (q_0 \text{ and } q_1 \text{ are calculated from } p_i \text{ and } p_{i+1}) \\ &= I(X, T, \text{segment i}) - I(X, 0, \text{segment i}) \end{aligned}$$



(a) Convolution (b) Heat diffusion (c) Ours

Figure 5: Distance functions computed using (a) convolution, (b) heat diffusion (with a grid size of 300x300 and a diffusion time of 20000) and (c) our d_2 function - computed in 0.0046s, 75.0918s, and 9.3727s respectively. Note the sharp convergence of hulls in the result of convolution, making it undesirable for our application. In comparison, heat kernel and our d_2 function give smooth hulls, but the computation of hulls using heat kernel is comparatively costly.

$I(X, t,$ segment i) is the primitive of $\|q_0 - tq_1\|^{-2}$ with respect to t, and T is the length of the segment.

When integrating, $\|q_0 - tq_1\|$ is the Euclidean distance between x and the point y along the segment. We rewrite $\|q_0 - tq_1\|^2$ as a quadratic function:

$$\|q_0 - tq_1\|^2 = t^2 + 2tb(x) + c(x)$$

which simplifies the integral calculation.

Expressing it as $(t + b(x))^2 + v(x)$ (with $v(x) > 0$), we can easily find the primitive of $\frac{1}{(t+b(x))^2+v(x)}$ and compute the integral. We denote this primitive by I.

Since the integral is linear, the integral of $\|x - y\|^{-2}$ over the broken line is the sum of the integrals over each segment:

$$\int_{\text{broken line}} \|x-y\|^{-2} = \sum_i \left(I(X, T, \text{segment } i) - I(X, 0, \text{segment } i)\right).$$

Thus, the final expression for the function d_2 of a segment is:

$$d_2 = \frac{\sqrt{A}}{\sqrt{\sum_i \left(I(X, T, \text{segment } i) - I(X, 0, \text{segment } i)\right)}}.$$

Using the linearity of the sum operator and applying the gradient operator, we get the gradient of function d_2:

$$\nabla d_2 = -\sqrt{A} \cdot \frac{\sum_i \left(\nabla I(X, T, \text{segment } i) - \nabla I(X, 0, \text{segment } i)\right)}{2\left(\sum_i I(X, T, \text{segment } i) - I(X, 0, \text{segment } i)\right)^{3/2}}$$

Figure 5 shows the hulls created using convolution surfaces [50] and the heat equation [12] on a polyline with 2000 vertices. Though much faster, with a running time of 0.0046 seconds, the convolution-based approach could not capture important features of the input curve. The solution using heat diffusion could satisfactorily capture the important features, but it took around 75.0918 seconds to compute. Moreover, the precision of heat diffusion heavily depends on the grid size, diffusion time, and chosen time step. With our explicit solution, we could get similar but precise results in 9.3727 seconds. Also, compared to the heat diffusion, our d_2 function is simpler and more efficient as it provides a closed-form expression for both the distance and its gradient at any given point; the heat diffusion

Figure 6: An illustration showing our d_2 function used for sketch stroke inflation.

requires running a simulation for every point on the grid, for a theoretically indefinite amount of steps. In contrast, our method does not require a discrete domain definition. Finally, the calculation of the d_2 distance and gradient is solely dependent (and linearly so) on the number of sections in the shape, making it especially well-suited for our application involving strokes and polylines.

Another alternative would be to work with kernel-based methods, but truncated kernels with a small time step cannot evaluate gradients far off the spine. However, large kernels require the use of an FFT to remain efficient, which has higher theoretical complexity. In addition, it is unclear whether kernels provide stable estimates for the gradient everywhere. For example, when a spine gets close to itself, a large time step might fuse structures numerically.

The versatility of the proposed explicit d_2 function extends beyond its immediate application in 3D modeling. For example, this d_2 function can be used in applications such as computing curve offsets [56], rasterization [33], animation [13], and vector art [39]. Figure 6 illustrates the result of a prototype that uses our d_2 function to inflate hand-drawn strokes. As can be seen, while the thickness increased, the sketch grew without merging nearby features. In addition, it is worth noting that the d_2 function possesses an important property that allows for incremental updates when a segment of the spine is moved, making it particularly useful for 2D animation applications.

3.3.2 Rib length Optimization. The previous section described how ribs grow following a gradient. We perform this iterative process and stop when an edge in the input image is reached. These edges stem from a Canny edge detector. Using all ribs directly might result in incorrect shapes due to edges generated by unwanted occlusions or noise in the input. To make the process more robust, we employ a rib length optimization algorithm, relying on symmetry constraints and edge information available on either side of the user-drawn spine as outlined in Algorithm 1.

Algorithm 1 Rib Length Optimization Algorithm

```
procedure RibLengthOptimize(Ribs R, d_Max)
    Candidates = ∅
    for 100 iterations do
        Let random R_S ⊂ R with ∀r in R_S: length(r) < d_Max
        for g ∈ {R − R_S} with length(g) < d_Max do
            if Penalty(R_S ∪ {g}) < 1.0 then
                R_S ∪ {g}
        if SIZE(R_S)> Threshold then
            Candidates = Candidates ∪ R_S
    for c ∈ Candiates do
        c = completeViaInterpolation(c)
    return argmin_{c∈Candidates} Penalty(c)
```

Figure 7: Superposition of the gradient curves of a user-drawn spine and the blurred detected edges used to evaluate the proximity of a point to a detected edge. Intersections are highlighted in red. The algorithm tries to choose as many points as possible in these red areas while satisfying other smoothness constraints.

The algorithm selects a set of ribs with a minimal penalty (defined below), and its endpoints are then interpolated to produce ribs for gradient curves that were not selected. In case the candidate list is empty, we would have to restart the function with an increased maximum distance d_{Max}. In practice, the algorithm can be implemented without a candidate list but by tracking a minimum. Similarly, sorting the ribs by length would make the selection according to d_{Max} very simple.

Penalty Energy. We define the penalty energy as the simple sum of three terms:

- *Distance Penalty:* This penalty ensures that the rib endpoints stay close to the image edges. To compute this, we first blur the Canny edge image with a normalized box filter (kernel size: 1% of the image width - resulting in a figure similar to Figure 7). Depending on the pixel color p at the endpoint of the rib, we define the distance penalty as 0 if the pixel color is between 200 and 255 (close to the edges), or $1 - p/200$ otherwise (pixel far from the edges).
- *Neighbor Penalty:* This penalty ensures that neighboring ribs have a similar length (distance to the spine) - or in other words, maintains consistency between adjacent ribs. Let the ribs be separated by distance s on the hull H_0, for corresponding steps Z_1 and Z_2 along the ribs, while following the gradient, the penalty energy is defined as $3 \times max(0, |d_2(Z_1) - d_2(Z_2)|/s - 1/4)$. This penalizes large differences in rib lengths relative to their separation, ensuring smooth transitions between neighboring ribs.
- *Opposite Penalty:* This penalty ensures that the ribs have their respective step points located at similar distances w.r.t. the spine (trying to maintain symmetry). Given the associated points Z_1 and Z_2 from the stepping points on both sides of the spine, the penalty energy is defined as $0.25 \times |d_2(Z_1) - d_2(Z_2)|/d_2(Z_1)$.

In case no direct neighboring or opposite ribs exist, we interpolate Z_2 from the nearest already added neighboring ribs. The trade-off between these penalty measures was empirically chosen. The effect of the penalty criteria are shown in Figure 8. As can be seen, the rib-length optimization on distance criteria alone resulted in ribs that are jutting out of shape on one side (Figure 8(b)), as the algorithm tried to find a smooth solution where edges were missing (joint between the hind leg and tail). Having constraints

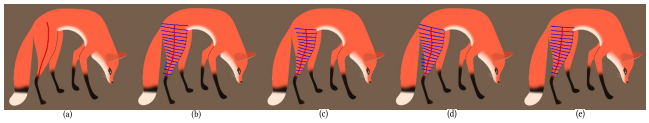


Figure 8: Effect of different terms in the penalty energy. (a) User-drawn spine (in red color - please note that this is the only user interaction in this example), (b) Penalty energy with only distance to edge, (c) with distance to edge and similarity to previous rib, (d) with distance to edge, similarities to previous and next ribs, (e) with all our penalty criteria (Image courtesy: PixaBay).

on neighboring ribs leads to stretched or contracted ribs (Figure 8(c-d)). Once we used all the penalty criteria, as demonstrated in Figure 8(e), we could get the desired rib structure.

The result of using this rib optimization can be seen in Figure 9. It can handle not only missing edges but also noisy boundaries (typical when the user chooses the edge detection option over a natural image). The efficiency of our rib length optimization on a sample input is shown in Figure 10. Our solution automatically generated a decent set of ribs despite the presence of noise and missing data near and around the beak. Even if the automatically computed ribs do not match the user's expectations, they can be easily edited, as explained next. Figure 10(f-g) shows a result after rib adjustment.

3.4 Lofting

The final step of conversion transforms the rib-spine combination into a 3D mesh based on a provided cross-section. The cross-section can be chosen from a predefined set (containing simple shapes, such as circles, rectangles, triangles, etc.) or sketched out by the user.

Once a cross-section is provided, the spine is first centered by taking the midpoint along opposing ribs. The center of the cross-section is then aligned with this midpoint (Figure 11) and scaled to match the length of the ribs. Consequently, the shape's borders will coincide with the rib endpoints and, thus, with the edges detected in the input image. The cross-section is then rotated and connected with the cross-sections corresponding to the neighboring ribs. If there is no neighbor, we triangulate the interior of the cross-section to create a closed shape. Because the distance function is smooth and the spine is centred with respect to the ribs, robustness is increased, and a certain imprecision in the user annotations is acceptable (see Figure 12).

The process is very fast and fluid in terms of interaction, as the mesh is generated swiftly. Upon sweeping the spine, each pair of ribs adds a new boundary piece to the 3D shape until it is complete.

4 User Interaction

The main interactions made available by our system, as demonstrated in Figure 13, include:

- Rib editing: The presence of large occlusions or noise typically ends up in ribs that do not match the user's expectation, or sometimes the user uses the photograph just as a guide and has to locally update the shape. In such cases, our interface allows the users to directly manipulate the rib endpoints

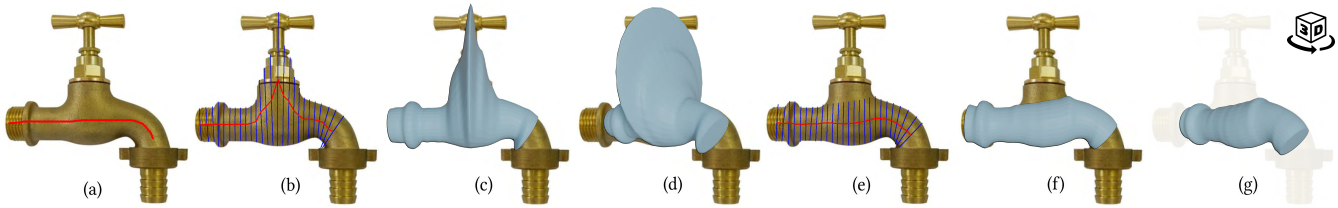


Figure 9: Effect of our rib length optimization. (a) User-drawn spine (in red color) (b) Spine-rib system automatically generated using our system without rib length optimization, (c-d) Corresponding 3D model, (e) Spine-rib system automatically generated using our system with rib length optimization, (f-g) Corresponding 3D model.

in two ways: either by simply dragging and dropping a rib endpoint or by drawing strokes - to which the nearby ribs will grow or shrink, thus, adjusting their lengths.

- Spine reposing: The operation that allows the reorientation of a spine (e.g., to match a part to the rest of a created object).

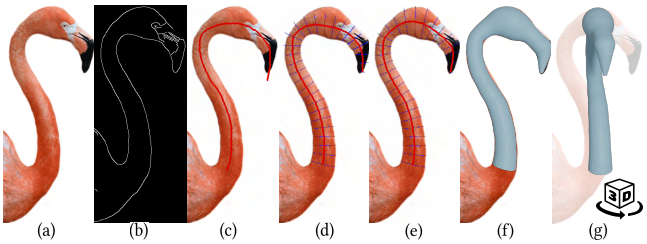


Figure 10: Effect of rib length optimization on input with noisy edges. (a) Input image, (b) Result of Canny edge detection, (c) User drawn spine (in red color), (d) Spine-rib system generated using our method, (e) Spine-rib system after editing, (f-g) Final 3D model generated by our method.

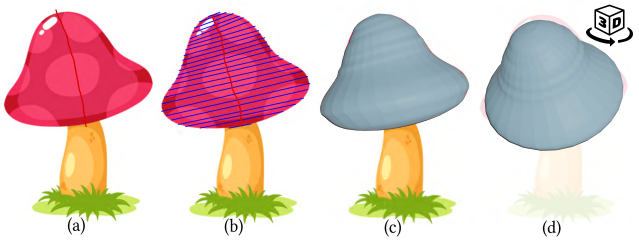


Figure 11: User-drawn spine (a), spine rearranged after computing ribs (b), and corresponding 3D model (c-d).

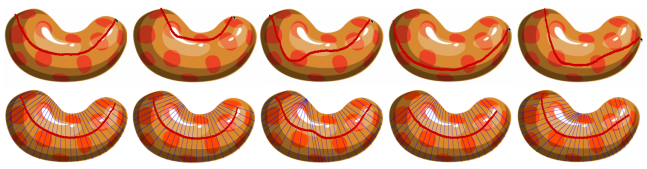


Figure 12: Top: User-drawn approximate spines, Bottom: Ribs and updated spines computed by our system. The original image is taken from PixaBay.

Here, the user selects a spine and draws a new stroke to which the spine is aligned. Specifically, the user-drawn stroke will be considered a new spine, but instead of computing gradient-edge intersections and then applying the rib-length optimization, we copy the ribs from the original reference spine, scaled by the relative stroke length.

- Cross-section editing: During the lofting phase, our interface provides a dictionary of common cross-section shapes. In addition, a user can sketch and define custom cross-sections.

The supplementary video provides a demonstration of these interactions in use.

5 Results and Discussion

Several results generated with SpineLoft using sketches (taken as bitmaps) and photographs as reference images can be seen in Figures 14 and 15. In several examples, object parts are generated and composited (Figure 16) using several spines to define parts, and defining appropriate cross-sections enables the creation of complex objects. It has to be noted that SpineLoft supports shapes, which do not lend themselves well to inflation or approximation by generalized cylinders. Please note that the results can be further smoothed as post-processing.

5.1 Comparison of Functionalities

In this section, we compare various key features of SpineLoft to existing work and summarize the findings in Table 1.

- Type of input - Are general images supported as input? Many sketch-based modeling methods, e.g., [1], [23], [7], [51], require an input sketch, whereas, SpineLoft uses images as input.
- Ability to select parts - Can a user pick and selectively model parts of an object? Inflation-based methods such as Ink-and-Ray [55] and NaturaSketch [40] inflate complete boundaries and lack a clear part definition. CreatureShop [61] and Andre et al. [1] also require the user to define required boundaries explicitly. In contrast, SpineLoft provides the freedom to select the required parts alone.
- Editable - Is the resulting shape directly editable? Only RigMesh [7], MonsterMash [16], and Ours have this functionality. RigMesh enables modifying a 3D pose, which is somewhat reflected by our method's spine reposing. In addition, we support local edits like MonsterMash. Thanks to

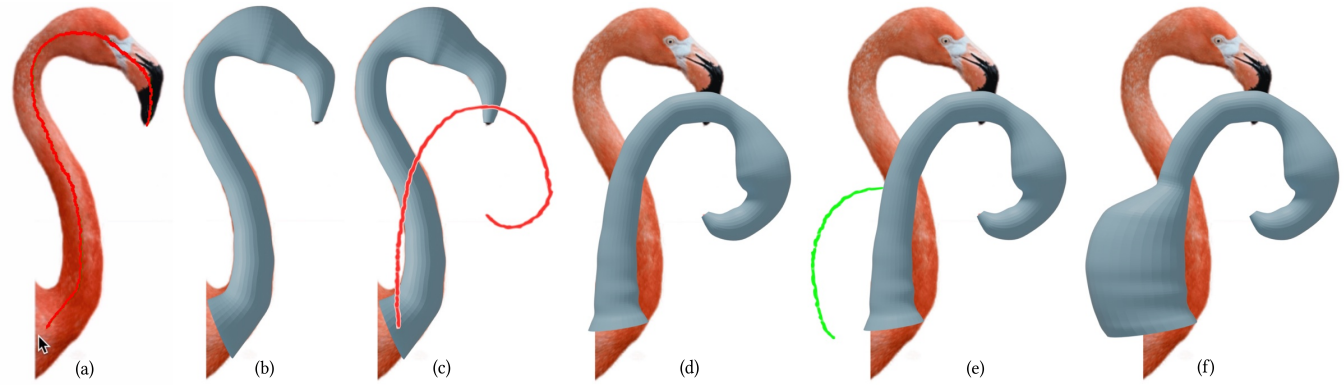


Figure 13: Spine reposing and rib deformation operations. (a) User-drawn spine in red color, (b) Corresponding 3D model, (c) New user-drawn stroke in red color for reposing, (d) Resulting reposed 3D model, (e) User-drawn stroke in green color for deforming spines, (f) Resulting 3D model.

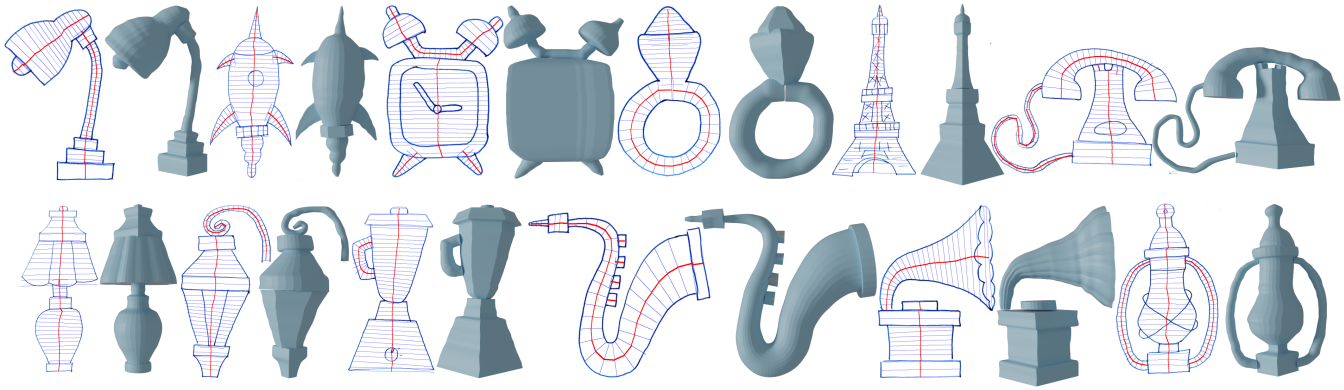


Figure 14: Various results generated by our interface on sketch inputs. Each tuple shows the input sketch (along with the spine-rib systems) and the resulting models.

Figure 15: Various results generated by our interface on image inputs. Each tuple shows the input image and the resulting model overlayed on the appropriate part of the image. To illustrate the flexibility of our approach, we used low-polygon cross-sections. Images are taken from PixaBay.

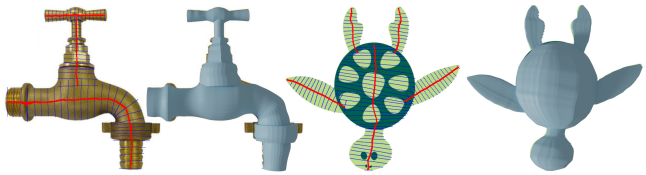


Figure 16: Models generated by combining multiple spine-rib systems and appropriate cross-sections

the spine-rib system, the results of SpineLoft can be easily edited.

- Arbitrary cross-section - What cross-sections can be used? Many existing methods use circular inflation, which results in blobby shapes. NaturaSketch and 3-Sweep [10] define a particular and limited set of cross-section choices. Andre et al. [1] uses arbitrary cross-sections but requires them to be drawn from a fixed viewpoint. SpineLoft enables arbitrary cross-sections.
- Requires a clean boundary - Can noise and missing edges be handled? 3-Sweep handles some small degree of missing/noisy boundaries but fails for larger occlusions and inaccuracies. Thanks to the rib length optimization algorithm, qualitative comparisons indicate that SpineLoft is more robust (see Figure 17).
- Riggable representation - Can the result be rigged? Though not made explicit, some methods could be similarly suited as ours, such as 3-Sweep, Gingold et al. [19], and a modified MonsterMash in the spirit of RigMesh.
- Precise input - How precise do user annotations have to be? Precise input is time-consuming and requires careful interaction. The use of the smooth distance function and the centering of the spine enables a degree of inaccuracy in the user scribbles. It has to be noted that this functionality is unique to SpineLoft.

5.2 Comparison of Results

We compared our results with those generated by 3-Sweep [10] and two variants of SpineLoft: Case 1 (with the d_∞ function and without rib editing) and Case 2 (with the d_2 function, without rib length optimization, and without rib editing), and are shown in Figure 17. We concentrate our comparison on 3-Sweep because it is the only method, like ours, that uses an image as input and is based on a spine-like stroke. For a more detailed comparison with other sketch-based modeling tools, please refer to the Appendix B.

The 3-Sweep method employs a sweeping technique that can handle minimal inconsistencies in the input image/sketch. However, it is difficult to control the sweeping when there is bending in the sweeping profile (as seen with the banana shape in Figure 17). Additionally, significantly missing edges present further difficulties (for example, the copter in Figure 17), and non-circular cross-sections often result in undesirable results (for a fair comparison, we used circular profiles for most examples). In comparison with our easily editable spine-rib system, the 3-Sweep method offers more limited editing capabilities for the extracted 3D objects. It should be noted that compared to our solution, 3-Sweep can better handle open boundaries in sketches, as demonstrated in Figure 18.

In contrast, using the d_∞ function resulted in self-intersecting meshes (evident in the blobby shape and banana examples in Figure 17), while the d_2 function without rib length optimization and rib editing produced erroneous shape boundaries (as seen in the jar and copter examples in Figure 17).

5.3 Limitations

While the SpineLoft is effective in easily creating a variety of shapes, it lacks the flexibility to model complex geometries. These limitations arise mainly due to the fact that SpineLoft essentially creates a loft surface along a single 2D spine with orthogonal ribs. As shown in Figure 19, some shapes that it cannot create include :

1. Multi-curvature surfaces - As SpineLoft computes ribs as line segments orthogonal to the spine, it cannot represent surfaces with complex curvatures in multiple directions, such as hyperbolic paraboloids (e.g., the shape shown in Figure 19(a)), as this requires simultaneous positive and negative curvatures in different directions instead of simple orthogonal ribs.
2. Rotational interpolation - As the ribs are in 2D and are orthogonal to the spine, it cannot create twisted structures like pasta shapes (e.g. the shape shown in Figure 19(b)). Generating such a shape with high torsion or non-linear twist would require ribs to rotate along the spine and, hence, require complex interactions and expertise, which our current system does not support.
3. Non-uniformly scaled objects - As in other sweeping-based interfaces (e.g., 3-sweep [10]), our method is not designed for non-uniform scaling along the spine. Though SpineLoft allows varying rib sizes along the spine, complex non-uniform scaling operations cannot be performed using the current interface, making it difficult to model objects like toothpaste shown in Figure 19(c) - whose cross-section transforms from a circle to an ellipse along the spine.
4. Shapes with non-planar spines - To make the interactions accessible to novice users, we assume that the spines are in 2D, making it difficult to generate 3D shapes like helical structures (for e.g. the shape shown in Figure 19(d)) which requires a 3D spine.

In addition to the shapes it can generate, our current interface implementation has two minor shortcomings: it trims the user-drawn scribbles on both ends while computing normals, which leads to users drawing scribbles slightly longer than needed. In addition, we do not add caps to the generated objects (Figure 3). One could always close the shapes by trimming the appropriate ribs if desired.

5.4 Preliminary User Evaluation

We conducted a user evaluation of SpineLoft through three distinct studies, each targeting different aspects of the system. The first study focused on novice users with little to no prior 3D modeling experience to evaluate the usability and intuitiveness of the interface. The objective was to measure the learning curve and initial user experience of novice users. The second study focused

Method	Properties						
	Image as input?	Ability to select parts?	Editable ?	Arbitrary cross-section?	Require a clean boundary?	Riggable?	Precise input?
Teddy [23]	No	NA	No	No	Yes	No	NA
Gingold et al. [19]	As Ref	Yes	No	No	NA	Yes	Yes
Andre et al. [1]	No	Yes	No	Yes*	Yes	No	Yes
NaturaSketch [40]	As Ref	No	No	Yes*	Yes	No	NA
RigMesh [7]	No	NA	Yes*	No	Yes	Yes	NA
3-Sweep [10]	Yes	Yes	No	No	Yes	Yes	Yes
Snapping [51]	No	Yes	No	No	Yes	No	Yes
Ink-and-Ray [55]	No	No	No	No	Yes	No	NA
MonsterMash [16]	As Ref	NA	Yes*	No	Yes	Yes	Yes
CreatureShop[61]	As Ref	Yes	No	No	Yes	No	Yes
Ours	Yes	Yes	Yes*	Yes	No	Yes	No

Table 1: Comparison of different methods. Many works support images as a reference (marked as "As Ref"), but only 3-sweep and SpineLoft are designed to benefit algorithmically. Only our method is able to handle images with incomplete contours or occlusion (compare Figure 2).

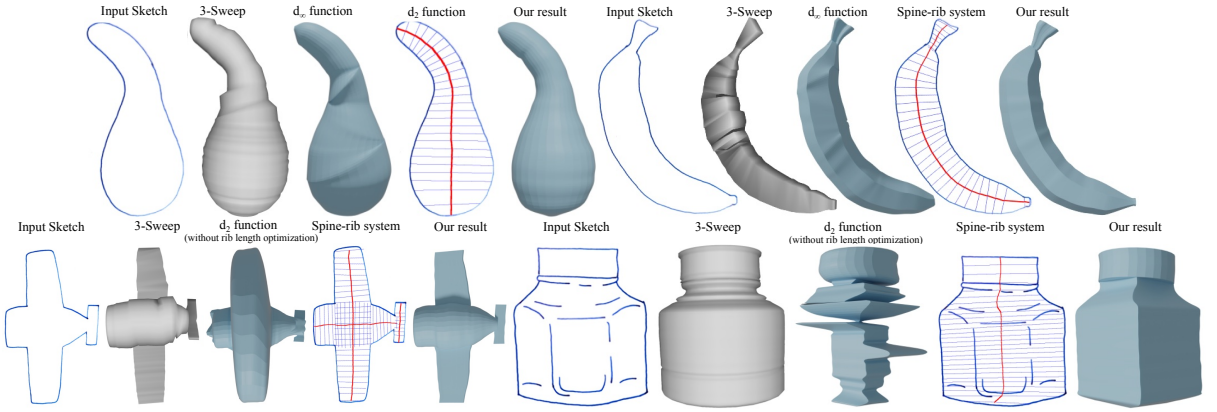


Figure 17: Comparison of our method w.r.t. 3-Sweep, d_∞ function and d_2 function without rib length optimization.

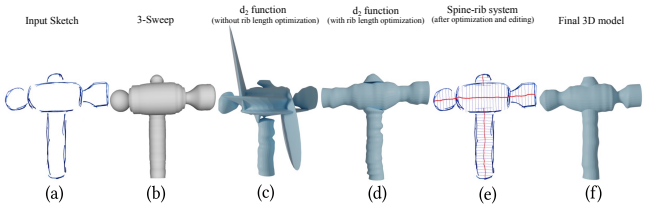


Figure 18: An example of a sketch drawn with open and multiple strokes where 3-sweep works better. (a) Input sketch, (b) Result of 3-sweep, (c) Our result without rib length optimization, (d) Our result with rib length optimization, (e) Spine-rib system after editing, (f) Our final result.

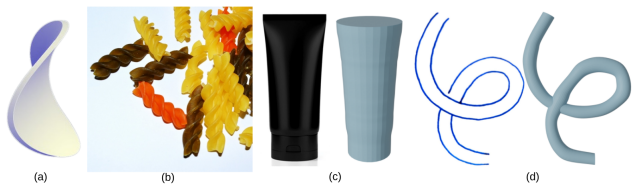


Figure 19: A few representative failure cases of our system. Our system cannot model (a) Saddle-shaped surface with double curvature, (b) Shapes with twisted/rotating profiles, (c) Shapes with non-uniform scaling, and (d) Helical structure from a single sketch.

on obtaining in-depth feedback from experienced users about the capabilities and limitations of the system compared to existing professional 3D modeling tools. The third study focused on users with varying levels of modeling experience and was intended to explore the creative potential of the system. The details of the study are as follows:

Novice user study: Ten participants aged between 12 (with parent's consent) and 43 with little to no prior experience in 3D modeling volunteered to try our system. The users were shown the video

of our demo (as in the supplementary video) and were allowed to familiarize themselves with our system for 10-15 minutes. After this phase, they were asked to complete three tasks of increasing complexity: creating a cylinder (from a simple image of a rectangle), a bird neck (using the reference image shown in Figure 1) and then a mug from an image with occlusion (using the reference image shown in Figure 2). We measured the task completion times and conducted a post-study usability survey on 1-5-point Likert scale. The survey focused on various aspects of our system - task understanding, user-friendliness, feeling of control, task completion efficiency, helpfulness of spine-rib system, intuitiveness of spine drawing, and satisfaction with the final 3D shape.

The results of the survey were highly encouraging. The mean scores across all the usability metrics ranged from 4.0 to 4.5 (with an average magnitude of deviations from the mean: 0.4 to 0.64) - suggesting that our interface is intuitive and user-friendly for beginners. In addition to the usability metrics, we also included questions to understand the overall experience and intentions for future engagement. The questions were about the enjoyment of the user while using the system, future use for creative tasks, confidence in using the system and the user's willingness to create more models. The responses to these questions were also encouraging, with mean scores varying from 4.3 to 4.5 (with an average magnitude of deviations from the mean: 0.5 to 0.56). These high scores, especially for enjoyment and intention for future use, were particularly promising as they suggest that our system effectively engages novice users and develops their interest in 3D modeling activities. It is also worth noting that the average modeling time for cylinder, bird neck and mug were 67s, 71s, and 162s, respectively - demonstrating the ability to quickly create 3D models. To gain a deep understanding of user perception, we also asked the participants two open-ended questions: "What did you like most about the system?" and "What further improvements would you suggest?". A thematic analysis of the answers to the question "What did you like most about the system?" reaffirmed various strengths of our system:

- Intuitiveness - users appreciated the ability to create 3D models from 2D images with simple inputs.
- Editable ribs - the ability to manipulate ribs for fine-tuning 3D shapes was frequently mentioned as a positive feature.
- Ease of use for novice users - many participants, especially those doing 3D modeling for the first time, found the system accessible and enjoyable.
- Spine-rib metaphor - users found the spine drawing and rib editing metaphor intuitive and useful for creating 3D models.

Participants also provided suggestions for future improvements, including the recommendation to add color-coded feedback for different modes (for example, a different color for ribs that will get affected while deforming ribs) and an improved rib computation to reduce the required edits and the time.

Expert user study: To gain insights from an experienced user point of view, we asked four experts with over two years of 3D modeling experience to evaluate our system. They were shown the demo of our system and asked to model the faucet shown in Figure 16. In addition, in the second part, they were asked to edit the faucet to modify the shape as they wanted. The experts successfully recreated the model in less than 3 minutes and could easily modify it to match their imagination. Once satisfied with the modeling, they provided valuable qualitative feedback about the system. Thanks to the ability to model directly from a reference image and the easy-to-edit spine-rib representation, all of them unanimously agreed that Spineloft would be a compelling alternative to their current preferred 3D modeling software - ranging from Blender to Autodesk Inventor.

- The workflow in itself seems pretty innovative. It would be nice to use it for prototyping but not for very precise modeling.
- It would be nice to have an automatic merging of individual parts and an option to edit the ribs long after its creation, whereas the current system, after creating a new spine, makes the previous 3D model uneditable.
- It would be nice to have it as a plugin for some software, such as Blender, so that I can build over the prototypes I create.
- Having an option to manually add or delete ribs would be beneficial (especially while using it for CAD modeling).

Study exploring creative potential: Our third user study explored the creative potential of the system. To round up our modeling tool, we added simple inflation tools - using Delaunay inflation [43] - and planar-sheet extrusion to craft elements like spheres, antlers, and wings. The composition of all parts created is done via Meshmixer [49], which fuses the components.

We tested SpineLoft with 12 users aged between 15 (with parent's consent) to 46 years, of which only two had some prior modeling experience. We showed them the demo of our system and allowed them to explore it for 30 minutes. After that, we asked them to model some imaginary characters by mixing and matching parts from different images. Figure 20 shows a few models they created, and it took 10 to 20 minutes for them to create the complete model (including the time for spatial arrangement and web-searching for the appropriate images). After each modeling session, we collected feedback from the users about the overall modeling experience. The feedback was overall positive, and the obtained fast prototyping results illustrate the strength of our solution. Users mentioned that "it is easy and enjoyable" to work with the system and that "the entire process was a lot of fun". In several cases, especially the inexperienced users were surprised that they "had complete control" and "could do whatever I want".

We also asked the users to rate the interface based on the overall experience and the fun they had during the modeling process, from Very bad to Excellent. All the users rated it as Very Good or Excellent and unanimously gave positive feedback, such as: "the entire process was a lot of fun, and we enjoyed it a lot".

In conclusion, our user studies indicate that SpineLoft provides an intuitive framework for 3D modeling. The positive feedback across all three studies suggests that our approach has the potential to lower the entry barrier for 3D modeling, making it accessible and enjoyable for novice users.

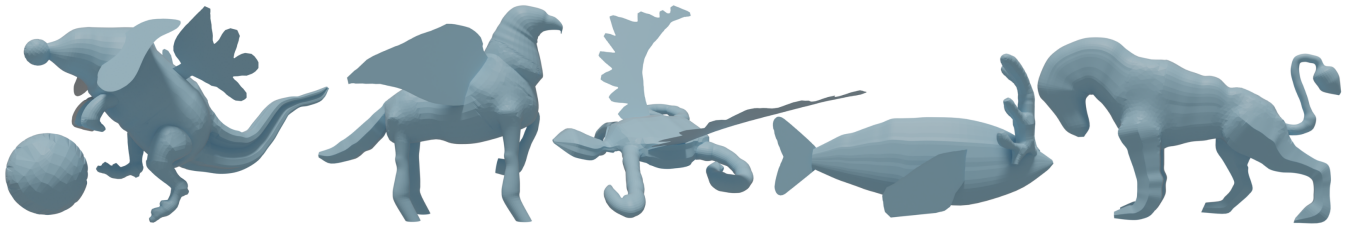


Figure 20: Some imaginary characters modeled using our interface by novice users (without prior modeling or designing experience) during the user study in less than 20 minutes (including the time for searching and finding appropriate images, drawing sketches wherever required, modeling parts using our interface and assembling them together).

5.5 Future Work

We envision future work in two primary directions. The first focuses on improving the current user interface to provide an enriched set of modeling options - making SpineLoft more suitable for intermediate/expert-level users. This includes implementing 3D rotational interpolation for cross-sections, enabling the creation of 3D cross-sections to model complex surfaces, and developing a more sophisticated rib length optimization framework. In addition to this, the ability to model hollow objects, such as the interior of the mug shown in Figure 2, could be envisioned. As typically done in Constructive Solid Geometry (CSG) modeling, this could be easily done by adding a mesh difference operation, which subtracts one solid from another.

The second direction involves extending the system into full 3D space. This includes developing a system similar to Skippy [29] to sketch 3D spines from a 2D view, facilitating the generation of complex 3D shapes like helices. This extension to 3D interactions would allow editing of spines, ribs, and cross-sections in 3D, making it possible to create a variety of shapes, including those with non-uniform scaling profiles.

Additionally, developing a plugin of SpineLoft for established 3D sculpting platforms like ZBrush or Blender would enable users to leverage SpineLoft for rapid abstract shape creation, which can then be refined using the advanced tools available in these sculpting systems - enhancing productivity for artists and designers in various fields.

6 Conclusion

We introduced a simple yet powerful, interactive spine-rib-based solution, SpineLoft, allowing users to create 3D models from sketches or images rapidly. The proposed method uses a novel d_2 function and a rib length optimization algorithm to create easily editable ribs from a user-drawn approximate spine. The proposed method is easy to use for novice users, as it does not require perfect precision. It helps develop rapid prototypes and base meshes (which can be refined further using specialized tools like Zbrush). The user study confirms that the proposed method is accessible even for first-time users and enables them to generate complex models (which previously they never knew they could) in a fun and playful manner. Finally, our specialized distance function, which can be easily computed in an explicit way, can open up avenues for applications beyond shape modeling, such as vectorization and animation.

Acknowledgments

The authors would like to thank all the anonymous reviewers for their constructive comments, all the participants of our study, and Aurèle Boquet for his help with the derivation. The research is partially funded by the ANR JCJC project SketchMAD (ANR-23-CE33-0009), Immersive Tech Lab within Convergence AI at TU Delft, JST AdCORP (JPMJKB2302) and generous support from Adobe. The images used in this work were obtained from Pixabay (http://pixabay.com), a platform providing royalty-free images available for download and use under their standard licensing terms.

References

[1] Alexis Andre and Suguru Saito. 2011. Single-View Sketch Based Modeling. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling* (Vancouver, British Columbia, Canada) *(SBIM '11)*. Association for Computing Machinery, New York, NY, USA, 133–140.

[2] Adrien Bernhardt, Adeline Pihuit, Marie-Paule Cani, and Loic Barthe. 2008. Matisse: Painting 2D regions for Modeling Free-Form Shapes. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, Christine Alvarado and Marie-Paule Cani (Eds.). The Eurographics Association.

[3] Mikhail Bessmeltsev, Will Chang, Nicholas Vining, Alla Sheffer, and Karan Singh. 2015. Modeling Character Canvases from Cartoon Drawings. *ACM Trans. Graph.* 34, 5, Article 162 (nov 2015), 16 pages.

[4] Mikhail Bessmeltsev, Nicholas Vining, and Alla Sheffer. 2016. Gesture3D: Posing 3D Characters via Gesture Drawings. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2016)* 35, 6 (2016).

[5] Sukanya Bhattacharjee and Parag Chaudhuri. 2020. A Survey on Sketch Based Content Creation: from the Desktop to Virtual and Augmented Reality. *Computer Graphics Forum* 39, 2 (2020), 757–780.

[6] Alexandra Bonnici, Alican Akman, Gabriel Calleja, Kenneth P. Camilleri, Patrick Fehling, Alfredo Ferreira, Florian Hermuth, Johann Habakuk Israel, Tom Landwehr, Juncheng Liu, and et al. 2019. Sketch-based interaction and modeling: where do we stand? *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 33, 4 (2019), 370–388.

[7] Péter Borosán, Ming Jin, Doug DeCarlo, Yotam Gingold, and Andrew Nealen. 2012. RigMesh: Automatic Rigging for Part-Based Shape Modeling and Deformation. *ACM Trans. Graph.* 31, 6, Article 198 (nov 2012), 9 pages.

[8] Philip Buchanan, R. Mukundan, and Michael Doggett. 2013. Automatic Single-View Character Model Reconstruction. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling* (Anaheim, California) *(SBIM '13)*. Association for Computing Machinery, New York, NY, USA, 5–14.

[9] Renjie Chen, Ofir Weber, Daniel Keren, and Mirela Ben-Chen. 2013. Planar Shape Interpolation with Bounded Distortion. *ACM Trans. Graph.* 32, 4 (jul 2013), 12 pages.

[10] 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.* 32, 6, Article 195 (nov 2013), 10 pages.

[11] Xuejin Chen, Boris Neubert, Ying-Qing Xu, Oliver Deussen, and Sing Bing Kang. 2008. Sketch-Based Tree Modeling Using Markov Random Field. In *ACM SIGGRAPH Asia 2008 Papers* (Singapore) *(SIGGRAPH Asia '08)*. Association for Computing Machinery, New York, NY, USA, Article 109, 9 pages.

[12] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2013. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG)* 32, 5 (2013), 1–11.

[13] James Davis, Maneesh Agrawala, Erika Chuang, Zoran Popović, and David Salesin. 2006. A sketching interface for articulated figure animation. In *Acm siggraph 2006 courses*. 15–es.

[14] Fernando De Goes and Doug L James. 2017. Regularized kelvinlets: sculpting brushes based on fundamental solutions of elasticity. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–11.

[15] Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A. Efros, and Adrien Bousseau. 2018. 3D Sketching Using Multi-View Deep Volumetric Prediction. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1, Article 21 (jul 2018), 22 pages.

[16] Marek Dvorožňák, Daniel Sýkora, Cassidy Curtis, Brian Curless, Olga Sorkine-Hornung, and David Salesin. 2020. Monster Mash: A Single-View Approach to Casual 3D Modeling and Animation. *ACM Trans. Graph.* 39, 6 (2020), 12 pages.

[17] Even Entem, Loic Barthe, Marie-Paule Cani, Frederic Cordier, and Michiel van de Panne. 2015. Modeling 3D animals from a side-view sketch. *Computers & Graphics* 46 (2015), 221–230. Shape Modeling International 2014.

[18] Amelie Fondevilla, Damien Rohmer, Stefanie Hahmann, Adrien Bousseau, and Marie-Paule Cani. 2021. Fashion Transfer: Dressing 3D Characters from Stylized Fashion Sketches. *Computer Graphics Forum* 40, 6 (2021), 466–483.

[19] Yotam Gingold, Takeo Igarashi, and Denis Zorin. 2009. Structured Annotations for 2D-to-3D Modeling. *ACM Transactions on Graphics (TOG)* 28, 5 (2009), 148.

[20] Martin Guay, Marie-Paule Cani, and Rémi Ronfard. 2013. The Line of Action: An Intuitive Interface for Expressive Character Posing. *ACM Trans. Graph.* 32, 6, Article 205 (nov 2013), 8 pages.

[21] Tao Hu, Liwei Wang, Xiaogang Xu, Shu Liu, and Jiaya Jia. 2021. Self-Supervised 3D Mesh Reconstruction From Single Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 6002–6011.

[22] Matis Hudon, Mairead Grogan, Rafael Pages, and Aljosa Smolic. 2018. Deep Normal Estimation for Automatic Shading of Hand-Drawn Characters. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*.

[23] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. 1999. Teddy: A Sketching Interface for 3D Freeform Design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. 8 pages.

[24] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine-Hornung. 2014. Bounded Biharmonic Weights for Real-Time Deformation. *Commun. ACM* 57, 4 (apr 2014), 99–106.

[25] Rubaiat Habib Kazi, Tovi Grossman, Hyunmin Cheong, Ali Hashemi, and George W Fitzmaurice. 2017. DreamSketch: Early Stage 3D Design Explorations with Sketching and Generative Design.. In *UIST*, Vol. 14. 401–414.

[26] Yongkwan Kim, Sang-Gyun An, Joon Hyub Lee, and Seok-Hyung Bae. 2018. Agile 3D sketching with air scaffolding. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[27] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. 2023. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4015–4026.

[28] Vladislav Kraevoy, Alla Sheffer, and Michiel van de Panne. 2009. Modeling from Contour Drawings. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (New Orleans, Louisiana) *(SBIM '09)*. Association for Computing Machinery, New York, NY, USA, 37–44.

[29] Vojtěch Krs, Ersin Yumer, Nathan Carr, Bedrich Benes, and Radomír Měch. 2017. Skippy: Single View 3D Curve Interactive Modeling. *ACM Trans. Graph.* 36, 4, Article 128 (jul 2017), 12 pages.

[30] Mackenzie Leake, Gilbert Bernstein, and Maneesh Agrawala. 2022. Sketch-Based Design of Foundation Paper Pieceable Quilts. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–11.

[31] Thibault Lescoat, Maks Ovsjanikov, Pooran Memari, Jean-Marc Thiery, and Tamy Boubekeur. 2018. A Survey on Data-driven Dictionary-based Methods for 3D Modeling. *Computer Graphics Forum* 37, 2 (2018), 577–601.

[32] Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. 2018. Robust Flow-Guided Neural Prediction for Sketch-Based Freeform Surface Modeling. *ACM Trans. Graph.* 37, 6, Article 238 (dec 2018), 12 pages.

[33] Rui Li, Qiming Hou, and Kun Zhou. 2016. Efficient GPU path rendering using scanline rasterization. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–12.

[34] Yuwei Li, Xi Luo, Youyi Zheng, Pengfei Xu, and Hongbo Fu. 2017. SweepCanvas: Sketch-based 3D prototyping on an RGB-D image. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 387–399.

[35] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. 2004. Lazy snapping. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 303–308.

[36] Markus Lipp, Peter Wonka, and Pascal Müller. 2014. PushPull++. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–9.

[37] Zhongjin Luo, Jie Zhou, Heming Zhu, Dong Du, Xiaoguang Han, and Hongbo Fu. 2021. Simpmodeling: Sketching implicit field to guide mesh modeling for 3d animalmorphic head design. In *The 34th annual ACM symposium on user interface software and technology*. 854–863.

[38] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 1–15.

[39] Diego Nehab. 2020. Converting stroked primitives to filled primitives. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 137–1.

[40] Luke Olsen, Faramarz Samavati, and Joaquim Jorge. 2011. NaturaSketch: Modeling from Images and Natural Sketches. *IEEE Computer Graphics and Applications* 31, 6 (2011), 24–34.

[41] Luke Olsen, Faramarz F. Samavati, Mario Costa Sousa, and Joaquim A. Jorge. 2009. Sketch-based modeling: A survey. *Computers & Graphics* 33, 1 (2009), 85–103.

[42] Amal Dev Parakkat, Marie-Paule R. Cani, and Karan Singh. 2021. Color by Numbers: Interactive Structuring and Vectorization of Sketch Imagery. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*.

[43] Amal Dev Parakkat, Hair Hara Gowtham, Sarang Joshi, and Ramanathan Muthuganapathy. 2020. A digital assistant for shading paper sketches. *Visual Computing for Industry, Biomedicine, and Art* 3(15) (2020).

[44] Jianbo Peng, Daniel Kristjansson, and Denis Zorin. 2004. Interactive modeling of topologically complex geometric detail. In *ACM SIGGRAPH 2004 Papers*. 635–643.

[45] Mengqi Peng, Li-yi Wei, Rubaiat Habib Kazi, and Vladimir G Kim. 2020. Autocomplete animated sculpting. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 760–777.

[46] Sverker Rasmuson, Erik Sintorn, and Ulf Assarsson. 2020. User-guided 3D reconstruction using multi-view stereo. In *Symposium on Interactive 3D Graphics and Games*. 1–9.

[47] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. 2005. Design principles for tools to support creative thinking. (2005).

[48] C. Robson, R. Maharik, A. Sheffer, and N. Carr. 2011. Context-Aware Garment Modeling from Sketches. *Computers and Graphics* (2011), 604–613.

[49] Ryan Schmidt and Karan Singh. 2010. Meshmixer: An Interface for Rapid Mesh Composition. In *ACM SIGGRAPH 2010 Talks* (Los Angeles, California) *(SIGGRAPH '10)*. Association for Computing Machinery, Article 6, 1 pages.

[50] Andrei Sherstyuk. 1999. Kernel functions in convolution surfaces: a comparative analysis. *The Visual Computer* 15, 4 (1999), 171–182.

[51] Alex Shtof, Alexander Agathos, Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. 2013. Geosemantic Snapping for Sketch-Based Modeling. *Computer Graphics Forum* 32, 2 (2013), 245–253.

[52] Maria Shugrina, Wenjia Zhang, Fanny Chevalier, Sanja Fidler, and Karan Singh. 2019. Color builder: A direct manipulation interface for versatile color theme authoring. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–12.

[53] Sudipta N Sinha, Drew Steedly, Richard Szeliski, Maneesh Agrawala, and Marc Pollefeys. 2008. Interactive 3D architectural modeling from unordered photo collections. *ACM Transactions on Graphics (TOG)* 27, 5 (2008), 1–10.

[54] Lucian Stanculescu, Raphaëlle Chaine, Marie-Paule Cani, and Karan Singh. 2013. Sculpting multi-dimensional nested structures. *Computers & graphics* 37, 6 (2013), 753–763.

[55] 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, Article 16 (apr 2014), 15 pages.

[56] Wayne Tiller and Eric G Hanson. 1984. Offsets of two-dimensional profiles. *IEEE Computer Graphics and Applications* 4, 9 (1984), 36–46.

[57] Anton Van Den Hengel, Anthony Dick, Thorsten Thormählen, Ben Ward, and Philip HS Torr. 2007. Videotrace: rapid interactive scene modelling from video. *ACM Transactions on Graphics (ToG)* 26, 3 (2007), 86–es.

[58] He Wang, Kirill A. Sidorov, Peter Sandilands, and Taku Komura. 2013. Harmonic Parameterization by Electrostatics. *ACM Trans. Graph.* 32, 5, Article 155 (oct 2013), 12 pages.

[59] 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.* 32, 4, Article 123 (jul 2013), 15 pages.

[60] Mingliang Xu, Mingyuan Li, Weiwei Xu, Zhigang Deng, Yin Yang, and Kun Zhou. 2016. Interactive mechanism modeling from multi-view images. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–13.

[61] Congyi Zhang, Lei Yang, Nenglun Chen, Nicholas Vining, Alla Sheffer, Francis C.M. Lau, Guoping Wang, and Wenping Wang. 2022. CreatureShop: Interactive 3D Character Modeling and Texturing from a Single Color Drawing. *IEEE Transactions on Visualization and Computer Graphics* (2022), 1–18.

[62] Yue Zhong, Yulia Gryaditskaya, Honggang Zhang, and Yi-Zhe Song. 2020. Deep Sketch-Based Modeling: Tips and Tricks. In *2020 International Conference on 3D Vision (3DV)*. 543–552.

A Detailed derivation of the d_2 function

We define a natural generalized distance function between a point x and a polygon P, consisting of vertices P_0 to P_{k-1}, with perimeter $A = \sum_{i=0}^{k-1} \|P_{i+1} - P_i\|$.

The distance function of degree n between x and P is then defined as an integral on the contour of P [44]:

$$d_n(x,P) = A^{1/n}\left(\int_P \|x-y\|^{-n}\, dy\right)^{-1/n}.$$

When $n = 2$, it is evaluated as:

$$d_2(x,P) = \sqrt{A}\left(\int_P \|x-y\|^{-2}\, dy\right)^{-1/2}.$$

Calculating the integral directly would be costly. Having concentrated on one line segment of the polygon at a time, we can rewrite

$$\int_P \|x-y\|^{-2}\, dy \text{as } \int_{[p_{i+1},p_i]} \|x-y\|^{-2}\, dy$$

$$= \int_0^{\|p_{i+1}-p_i\|} \|x - \left(p_i + t\frac{p_{i+1}-p_i}{\|p_{i+1}-p_i\|}\right)\|^{-2}\, dt$$

Let $T = \|p_{i+1} - p_i\|$, $q_0(x) = x - p_i$, and $q_1 = \frac{p_{i+1}-p_i}{\|p_{i+1}-p_i\|}$. The equation can be expressed as:

$$\int_0^T \|q_0 - tq_1\|^{-2}\, dt = \int_0^T \left(\|q_0 - tq_1\|^2\right)^{-1}\, dt$$

With $a = \|q_1\|^2$, $b(x) = -2(q_0(x)\cdot q_1)$, and $c(x) = \|q_0(x)\|^2$, the equation can be rearranged as:

$$\int_0^T (at^2 + b(x)\cdot t + c(x))^{-1}\, dt.$$

Note that the polynomial $at^2 + bt + c$ is always greater than 0 and thus can be written as:

$$a^{-1}\int_0^T \left((t+u(x))^2 + e^2\right)^{-1}\, dt$$

with $u(x) = \frac{b(x)}{2a}$ and $e^2(x) = \frac{c(x)}{a} - u^2(x)$. Finally,

$$a^{-1}\int_0^T \left((t+u(x))^2 + e^2(x)\right)^{-1}\, dt$$

evaluates to:

$$\begin{cases} \left[\frac{-1}{a(t+u(x))}\right]_0^T = \frac{1}{a\cdot u(x)} - \frac{1}{a(T+u(x))} \\ \left[\frac{1}{a\cdot e(x)}\arctan\left(\frac{t+u(x)}{e(x)}\right)\right]_0^T = \frac{1}{a\cdot e(x)}\left(\arctan\left(\frac{T+u(x)}{e(x)}\right) - \arctan\left(\frac{u(x)}{e(x)}\right)\right) \end{cases}$$

It can be shown that (Pythagorean trigonometric identity)

$$a^2e^2 = \|q_1\|^2\|q_0\|^2 - (q_0\cdot q_1)^2 = \|q1 \times q0\|^2$$

Since a is supposed non zero (we exclude the case $p_i = p_{i+1}$), $e(x) = 0$ is true if and only if $p_0\vec{p}_1$ and $p_0\vec{x}$ are parallel, i.e. if x lies on the line formed by p_i and p_{i+1}.

We then derive the gradient of this distance function, in respect to x.

We define $I(x,t)$:

$$I(x,t) = \begin{cases} \frac{-1}{a(t+u(x))} & \text{if} \quad v(x) = 0 \\ \frac{1}{a\cdot e(x)}\arctan\left(\frac{t+u(x)}{e(x)}\right) = \frac{\arctan(w(x,t))}{\sqrt{av(x)}} & \text{if} \quad v(x) \neq 0 \end{cases}$$

with $v(x) = a\cdot e^2(x)$ and $w(x,t) = \frac{a(t+u(x))}{\sqrt{av(x)}}$ ($w(x)$ is only defined for $v(x) \neq 0$)

Hence, we have:

$$d_2(x) = \frac{A}{\sqrt{I(x,T) - I(x,0)}}, \nabla d_2(x) = -A\frac{\nabla I(x,\text{dist}) - \nabla I(x,0)}{2(I(x,\text{dist}) - I(x,0))^{3/2}}$$

Using the same notation, we can derive the gradients of the variables we use as:

$$q_0(x) = x - p_i \text{ ,so } \nabla q_0(x) = 1$$

$$b(x) = -2(q_0(x)\cdot q_1 \text{ ,so } \nabla b(x) = -2(\nabla q_0(x)\cdot q_1) = -2q_1$$

$$c(x) = q_0(x)\cdot q_0(x) \text{ ,so } \nabla c(x) = 2(q_0(x)\cdot \nabla q_0(x)) = 2q_0(x)$$

$$u(x) = \frac{b(x)}{2a} \text{ ,so } \nabla u(x) = -\frac{q_1}{a}$$

$$v(x) = c(x) - au^2(x), \text{ ,so } \nabla v(x) = \nabla c(x) - 2au(x)\nabla u(x)$$
$$= 2q_0(x) + 2q_1u(x)$$

$$w(x,t) = \frac{a(t+u(x))}{\sqrt{av(x)}} \text{ ,so } \nabla w(x,t) = \frac{a\nabla u(x)}{\sqrt{av(x)}} - \frac{a(t+u(x))\nabla(\sqrt{av(x)})}{av(x)}$$

Changing $\nabla u(x)$ and $\nabla v(x)$ by their expression, we get the following.

$$\nabla w(x,t) = -\frac{q_1}{\sqrt{av(x)}} + \frac{w(x,t)\nabla v(x)}{2v(x)}$$

In these calculations, t is the variable of the integrand and x is the position of the point on the image.

With this, we can finally derive the gradient of $I(x,t)$. The case $v(x) = 0$ is quite easy, we find:

$$\nabla I(x,t) = \frac{-q_1}{(a(t+u(x)))^2} \quad \text{if} \quad v(x) = 0$$

We now calculate $\nabla I(x,t)$ in the case $v(x) \neq 0$.

Let us define P and Q:

if $e(x) = 0$

if $e(x) \neq 0$ $P(x,t) = \arctan(w(x,t))$ and $Q(x) = \frac{1}{\sqrt{av(x)}}$

We have the following:

$$\nabla I(x,t) = Q(x)\nabla P(x,t) + P(x,t)\nabla Q(x)$$

With:

$$\nabla P(x,t) = \frac{\nabla w(x,t)}{1 + w^2(x,t)} \quad \text{and} \quad \nabla Q(x) = -\frac{a\nabla v(x)}{2(av(x))^{3/2}}$$

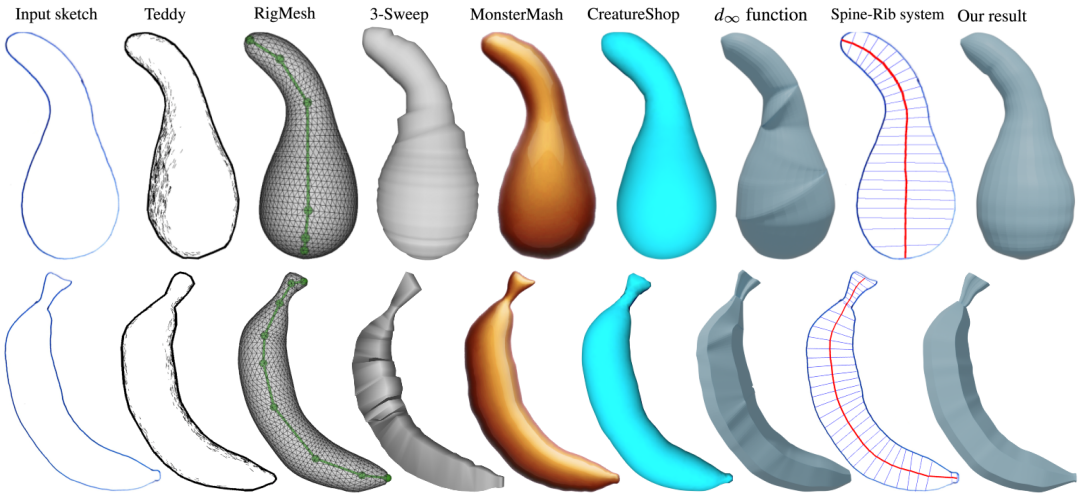


Figure 21: Comparison on a simple closed boundary (Top: Blobby shape) and a simple closed boundary with varying cross-sections (Bottom: Banana) Left to Right: Input sketch, Outputs of Teddy [23], RigMesh [7], 3-Sweep [10], MonsterMash [16], CreatureShop [61], d_∞ function, Our Spine-Rib system and the resulting 3D model

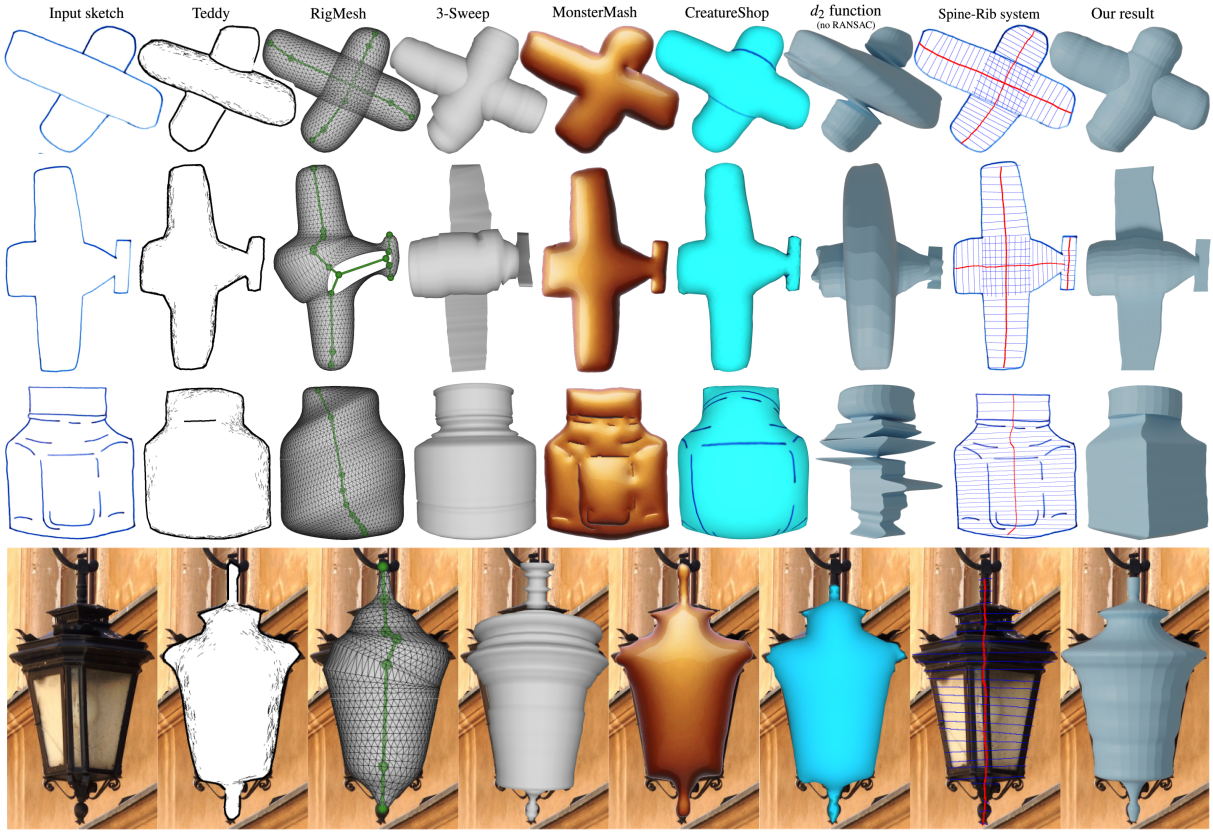


Figure 22: Top to Bottom: Comparison on an occluded boundary (Cross shape), object with missing boundaries (Plane), Noisy data (Noisy Jar) and Image (Streetlight); Left to Right: Input sketch, Outputs of Teddy [23], RigMesh [7], 3-Sweep [10], MonsterMash [16], CreatureShop [61], d_2 function without rib length optimization, Our Spine-Rib system and the resulting 3D model

$$\nabla I(x,t) = \frac{1}{\sqrt{av(x)}} \frac{\nabla w(x,t)}{1+w^2(x,t)} - \frac{a\nabla v(x)\arctan(w(x,t))}{(2(av(x)))^{3/2}}$$

This gives us the final expression of $\nabla I(x,t)$:

$$\nabla I(x,t) = \begin{cases} \frac{-q_1}{(a(t+u(x)))^2} & \text{if} \quad v(x) = 0 \\ \frac{1}{\sqrt{av(x)}} \frac{\nabla w(x,t)}{1+w^2(x,t)} - \frac{a\nabla v(x)\arctan(w(x,t))}{(2(av(x)))^{3/2}} & \text{if} \quad v(x) \neq 0 \end{cases}$$

As a reminder,

$$\nabla d_2(x) = -A\frac{\nabla I(x,\text{dist}) - \nabla I(x,0)}{2(I(x,\text{dist}) - I(x,0))^{3/2}}$$

B Comparison with other sketch-based modeling systems

In this section, we compare our results with those generated by five other methods (whose codes are available) and two variants of SpineLoft (Case 1: with d_∞ function and without rib-editing, Case 2: with d_2 function, without rib length optimization and without rib-editing). It should be noted that only 3-sweep is designed to work similarly to SpineLoft by taking images and user-drawn spine-like structures as input. All other methods are sketch inflation-based techniques that inflate a user-drawn closed boundary with little to no control over the shape, as they are not designed to take editability into account. To help readers understand how our methods differ from other sketch-based modeling tools, we list the main differences below.

- Teddy [23]: Teddy uses a simple inflation based on a constrained Delaunay triangulation. A drawback is the lack of support for image-based content. Instead, one needs to manually draw closed outlines. The results are typically blobby shapes. Missing boundaries or noisy sketches are not supported.
- RigMesh [7]: Similar to Teddy, RigMesh also creates blobby objects from user-drawn closed-curve sketches. It shares the drawbacks of Teddy, and complex curves with multiple branches do not lead to the desired result (compare plane in Figure 22). The same holds for the skeleton, which might differ from the expectations (noisy jar in Figure 22).
- 3-Sweep [10]: It uses a sweeping methodology and can handle minimal inconsistencies on the input image/sketch. Yet, it is difficult to control the sweep, especially when the profile requires bending (banana in Figure 21), and shape edits are locally not supported. Occlusion and missing edges also pose challenges, and non-circular cross-sections often lead to unwanted results (for a fair comparison, we used circular profiles for most results). It should be noted that compared to our solution, 3-Sweep can handle open boundaries in sketches, as shown in Figure 18.
- MonsterMash [16]: This approach uses blobby inflation, restricting the variety of possible results, but it does enable local boundary edits. Further, additional strokes can influence the inflation process (e.g., noisy jar in Figure 22). Nevertheless, it requires a closed boundary, making inputs, as in Figure 18, unsuitable.
- CreatureShop [61]: Following upon Teddy and RigMesh, it can actually handle images as inputs and relies on a Grabcut algorithm while the user traces the required boundaries as in Teddy/RigMesh. Arbitrary cross-sections or local editing are not supported.
- Our Method (Case 1): Our d_2 function results in smooth surfaces (especially when the spine is bent), while a d_∞ function, which we implemented for comparison, does not result in a suitable output. The blobby and banana examples in Figure 21 show the results generated using the d_∞ function.
- Our Method (Case 2): Using d_2, the importance of rib length optimization can be shown. To be fair, no edits were applied to the ribs. As can be seen in the cross, plane and noisy jar examples, the resulting ribs without rib length optimization extend to the region boundary (or hull in Figure 18), resulting in distorted and unexpected shapes.