Computer-Aided Design 128 (2020) 102904

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Descriptive: Interactive 3D Shape Modeling from A Single Descriptive Sketch[☆]

Cédric Bobenrieth[a,*], Frédéric Cordier[b], Arash Habibi[a], Hyewon Seo[a]

[a] ICube, Université de Strasbourg, France
[b] IRIMAS, Université de Haute Alsace, Mulhouse, France

ARTICLE INFO

Article history:
Received 28 October 2019
Received in revised form 28 February 2020
Accepted 9 June 2020

Keywords:
Sketch-based modeling
Descriptive sketch
Linear optimization

ABSTRACT

In this paper, we present a sketch-based modeler that reconstructs a 3D shape by combining a single descriptive sketch and minimal user intervention. The user provides a single 2D drawing in the form of a descriptive sketch, where solid curves describe the visible silhouette, and dashed curves the hidden outline. The curves are partitioned into a set of closed curves in a semi-automatic manner, each of which is consolidated into a closed surface element by solving a constrained optimization problem. The final 3D shape is generated by assembling these surface elements. The algorithmic reconstruction is complemented by allowing users to optionally guide the shape computation or correct any inaccuracy. This is done by successively specifying different kinds of local constraints on sparsely selected points in rotated views, such as adjustment of volume thickness along the projection line, or curvature discontinuity. Consequently, the range and complexity of shapes that can be created from a single-view sketch are significantly extended. We evaluate our solution by reconstructing a wide range of 3D models from sketches of various sources, and visually comparing the reference models and the shapes reconstructed by users.

1. Introduction

The common goal of sketch-based modeling (SBM) is to enable users to create 3D shapes from the sketch, an intuitive and efficient tool for visual communication. The user draws a set of strokes describing a 3D object on the sketching plane, which the modeler takes as input to compute a corresponding shape whose silhouette matches the input strokes. In solving the ill-posed problem, most modelers make use of human perceptual principles that are supposedly shared between the designers drawing the 2D sketches and the viewers who interpret them to mentally build the 3D shape. Leveraging our natural and well-trained skill for drawing and interpreting sketches, such modelers offer several advantages over conventional 3D modeling tools (3DS Max, Maya), among which is the ability to allow almost anyone to create 3D shapes easily and efficiently.

On the other hand, the shapes created with sketch-based modeling tools possess limited complexity and expressivity, as it becomes difficult to cover the full functionalities as offered by general modeling tools using solely a sketch-based interaction scheme. This paper addresses such difficulty and proposes solutions that combine a comprehensive sketching style borrowed from the descriptive geometry and minimal user intervention borrowed from conventional geometric modeling tools. The general goal of our modeler, which we name as Descriptive, is to expand the range and complexity of shapes that can be created from a single-view sketch.

The main limitation originates from the inherent ambiguities of a 3D shape reconstruction problem from a 2D drawing, i.e. there exists an infinite number of 3D shapes that match a given drawing. This problem becomes even more challenging when the drawing contains hidden parts that need to be estimated both in 2D and 3D. Although several existing methods have adopted perceptive or model-specific priors to make the problem solvable, it is still hard to solve it algorithmically since it involves the semantic understanding of the drawing. For example, the hidden part of a given shape depends on whether it represents a human or a plant. Our strategy to solve this problem is two-fold. First, the user sketches descriptive curves containing the description of hidden outlines in a dashed line style, which not only drastically reduces the ambiguity about the hidden shape (Fig. 1(a)) but also provides partial information on relative placements of enclosed shapes along the depth. Such a design choice is inspired by descriptive geometry, where hidden outlines are drawn by dashed curves. Next, the user can guide the automatic reconstruction by successively specifying local geometric constraints on sparsely selected curve points, through commonly accepted user interfaces as in conventional modeling

[☆] This paper has been recommended for acceptance by G. Giuseppe Patane.
[*] Corresponding author.
E-mail address: cedric-pc@hotmail.fr (C. Bobenrieth).

https://doi.org/10.1016/j.cad.2020.102904
0010-4485/© 2020 Elsevier Ltd. All rights reserved.

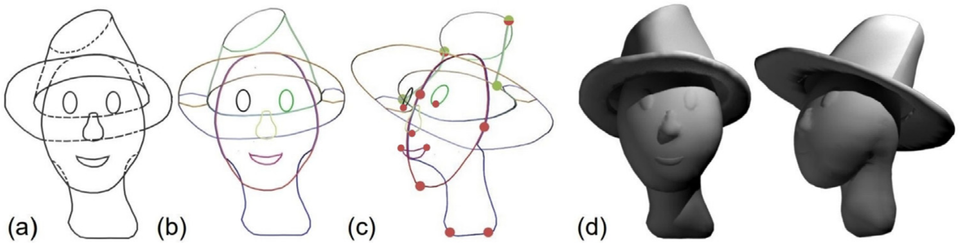


Fig. 1. Overview of the reconstruction process in our modeler. The descriptive curve drawing provided by the user (a), the user assembles the curve segments to generate the closed curves corresponding to the boundaries of closed surface elements (b). 3D positions of the curve points are computed (c) and the interpolating surface is generated (d).

tools. Such combination of a single-view sketch and minimum local control appears to be an efficient yet flexible way to convey the information on the intended shape.

Fig. 1 illustrates the modeling workflow with our system. The user starts by sketching the outline of an object including occluded parts, using the descriptive drawing style (Fig. 1(a)). The user is free to choose an arbitrary view, although there seems to be viewpoints preferred by designers that expose the multiple sides of the object and minimize the variation between a 2D projection and its 3D counterpart. The curve segments are then assembled into a set of closed curves (Fig. 1(b)), whose 3D counterparts are computed to allow the user to inspect the 3D shape to be reconstructed. The user can add local geometric constraints either to guide the initial reconstruction, or to modify it (Fig. 1(c)). In any case, a closed surface is generated for each curve, whose union is computed to result in a shape in such a way that both the projective constraint (sketch) and the user-specified local constraints are satisfied (Fig. 1(d)). We demonstrate that our strategies can significantly extend the modeling power of SBM while still maintaining the intuitive, easy-to-use interface. The remainder of the paper is organized as follows: Section 2 reviews related work. Section 3 explains the workflow followed by the user. Sections 4 and 5 respectively describe our surface generation process from the 3D curves, and the computation of the 3D coordinates of those curves. Section 6 presents results and discussion, and Section 7 conclusion.

2. Related work

Our work is related to reconstruction works from a single-view sketch or a photo. The reconstruction of relief from a 2D input rather than full a reconstruction is one way of coping with the missing depth information on the entire shape of the drawn object, which has been developed by several works such as [1,2], and [3]. Although our work aims at a more challenging problem of fully recovering the drawn shape, it shares some ideas with those works. As in our method, they first segment the input into a set of regions with the help of the user (either with annotation or interaction) and each region is assigned with a depth order. Volumetric shapes are then yielded by inflation. [2] add more interaction with the artist compared to [1] or [3]: the user annotates each region with different kinds of cues (slope and curvature) in order to refine the result of the inflation. In a sense, those interactions are very close to the way the user can refine the result in our method with the geometric constraints. However, these methods do not produce a full 3D model. Moreover, they are mainly dedicated to the reconstruction of organic shapes and have difficulties with polyhedral objects.

Most existing works employ the multi-view sketching framework, which is another way of coping with the missing information on depth and hidden parts. FiberMesh [4] produces an initial 3D shape from the first closed stroke. The user then may redraw curves on that shape in different viewpoints, which are used as control handles to smoothly deform the initial shape via functional optimization. [5] propose a modeler that enables the user to produce and refine a 3D shape by providing successive cross-sections of the intended shape through different orthotropic planes. In both works, complex shapes can only be produced if several sketches from different viewpoints are provided. BendSketch [6] adopts a substantially different approach, since they can produce fairly complex 3D shapes from a single annotated sketch. But in order to produce shapes other than Monge surfaces (height fields), again, several viewpoints are required. To our knowledge, Descriptive is the only modeler capable of producing such complex shapes with a single-view sketch.

Enhancement of the drawing in a more informative way is an idea that has been adopted by several existing modelers. By using a descriptive drawing style, the user provides more information than a common sketch that greatly helps the 3D reconstruction. [7] and [8] are based on the same principle and make use of designer sketches to reconstruct manufactured objects. Designers tend to describe the object's geometry by drawing pre-defined gesture curves to provide information on parallelism, orthogonality, etc. However, those methods are limited to the reconstruction of manufactured objects, and require the user to understand the designers' specific way of drawing. On the contrary, the descriptive style used in our modeler is accessible to everyone with some basic notion and experience, without requiring specific knowledge, and it can reconstruct a wide range of shapes, from organic objects to manufactured ones.

Restricting to shapes of common classes is a common idea of many methods, which make use of the common structure of a targeted class to resolve ambiguities. Naturally, they are designed to work well for the particular category of shapes such as buildings [9,10], organic shapes [11], clothes [12], or flowers [13], but seldom generalize to others. [10] use sketching interaction on the segmented 3D point cloud of a target building, in order to optimize the reconstruction workflow. [9] proposed another approach to combine sketching and procedural modeling for the reconstruction of urban buildings, by means of deep-learning based methods. The user provides a sketch separately for each component of a building to successively feed CNNs to find the closest snippet as well as its parametric model, which are merged together to form the final model. [14] demonstrate similar idea but applied to a broader scope of procedural models that are rather single, in contrast to composite models like buildings. Other than buildings, [15] present a method to create a 3D rigged character from a sketch by using the input 3D skeleton in the identical pose as the 2D drawing. The sketch is decomposed into individual parts of the character which will be used to produce 3D generalized surfaces of revolution which, together, form the final shape. However besides being restricted to characters, their method does not handle drawings with self-intersections. [16] propose a method for posing a given 3D character model via

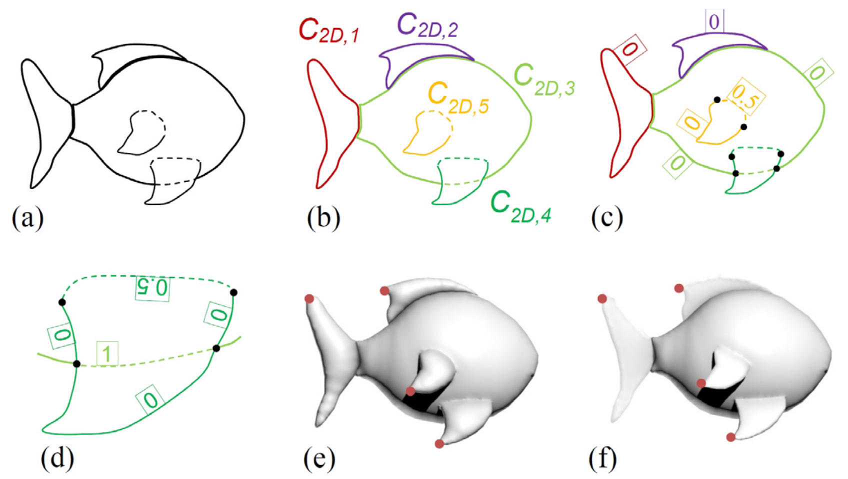


Fig. 2. The user provides an initial drawing. Occluded contours are drawn in dashed lines (a); then curve segments comprising the drawing are decomposed into a set of closed curves ($C_{2D,1}$ to $C_{2D,5}$) (b); the depth index is computed on each closed curve (c); a zoom of the pelvic fin to show the depth indices (c); optional depth constraints (shown with red dots) can be defined by the user on some selected points on the curve; closed surface elements are then created by inflating the surface patches generated from 3D closed curves (e), whose volume along z-axis can be adjusted by the user (f).

gesture drawing, a contour drawing of the posed character. Unlike the above works, Descriptive aims at reconstructing general surface geometry without any model prior, and solely from the user input, without any pre-existing model.

Closer to our work, [11] propose a method which automatically extracts from clean contour-line drawings the different parts that compose the final shape. The segmentation is performed automatically, and can fail when the input drawing does not have well-defined T-junctions or presents surface contact or cyclic arrangement between parts; such cases can be handled robustly in our modeler, since it is user guided. Moreover, their method is designed to work only on smooth organic shapes and thus does not produce the same variety of results as Descriptive. The optimization framework adopted in our work shares some similarity with the one proposed by [17], which allows the reconstruction of self-occluding objects from sketches. However, their algorithm can produce 3D objects with a circular cross-section, and therefore are limited to object with a tubular shape. Descriptive overcomes this limit and allows one to model general shapes, while handling self-occlusions well.

Recently, learning based methods have convincingly demonstrated their performance on the shape reconstruction tasks with sketch input. The inherent 2D to 3D ambiguity is solved by training neural networks using hundreds to millions of pairs of input sketches and the corresponding 3D shapes. Based on the deep convolutional neural network (CNN) which has been originally developed for image data, most of these works cast the problem as image-to-image translation, adopting the Encoder–Decoder networks. The sketch input is often augmented by preprocessed data prior to encoding, such as curvature flow [18] and selected feature points [19], or by additional sketches by the user from a rotated view [20]. The outputs of each network, the predicted shape in 2.5D (represented as voxel occupancy [20] or depth and normal maps [18,21]) from several viewpoints, are fused together to obtain the final shape. Although these CNN-based methods are promising, collecting the dataset and training the neural network is still expensive from the developer's point of view. From the user's perspective, our modeler can parse well the sketches including the occluded shape parts, and allows for the more precise, direct control over the generated shape. Subsequently, the user can generate more complex, sophisticated shapes, such as genus-n ($n \geq 1$) ('chair' in Fig. 19) or overlaid shapes ('clothed teddy bear' in Fig. 19).

3. User interface

The user interface of Descriptive has been developed in a way to make the modeling process as simple as possible while allowing the user to create a wide range of shapes. The user (1) starts by drawing the initial sketch, and (2) regroups the curve segments into a set of simple closed curves, which serve as boundaries of surface elements. Optionally, s/he can modify the computed shape in two successive steps: (3) constraint specifications on some selected curve points to guide its 3D reconstruction, and (4) modification of the volume of the computed 3D surface elements along the z-axis, to generate the final shape.

Initial drawing: The user draws the descriptive sketch from a single-viewpoint of an arbitrary choice, representing the silhouette curves or the edges (creases) of the shape (Fig. 2(a)). The user also draws the hidden curves in dashed or gray line, following the sketching style of descriptive geometry. This is done by toggling a button in the User Interface to specify that the drawn curves are hidden. Once the drawing is finished, a preprocessing stage is employed to generate a set of smooth B-spline curves from the drawing using the approach by [22]. The curves are then represented as a set of polygonal curves on the sketching plane ($z = 0$), after the discretization of the B-spline curves.

Partitioning the drawing into closed curves with depth indices: Given the initial drawing, our system splits the curves at their intersections, producing multiples of curve segments. The user manually groups these curve segments into a set of simple (i.e. with no self-intersection) closed curves, by selecting curve segments that collectively represent a closed curve. The result is a set of n simple closed curves $C_{2D} = C_{2D,1}, C_{2D,2}, \ldots, C_{2D,n}$ located in the sketching plane ($z = 0$) (Fig. 2(b)). Each curve segment is then assigned with a depth index in a semi-automatic manner. It encodes the number of surfaces lying between the viewpoint and the curve segment in 3D space, which is used to infer the correct depth order of 3D curves during the reconstruction steps. Initially, solid curve segments in the input drawing are given with the depth index 0, while dashed curve segments are assigned with 1. Curves that correspond to the junction where one element enters into another one are, later, automatically given a depth index composed of a decimal part of 0.5 (Fig. 2(c)). Depth indices can also be adjusted by the user when the computed depth order should be redefined. For instance, when two hidden curve segments intersect as in Fig. 3(a), the curve segment located behind the other one may be assigned with a depth order 2 (Fig. 3(d–e)),

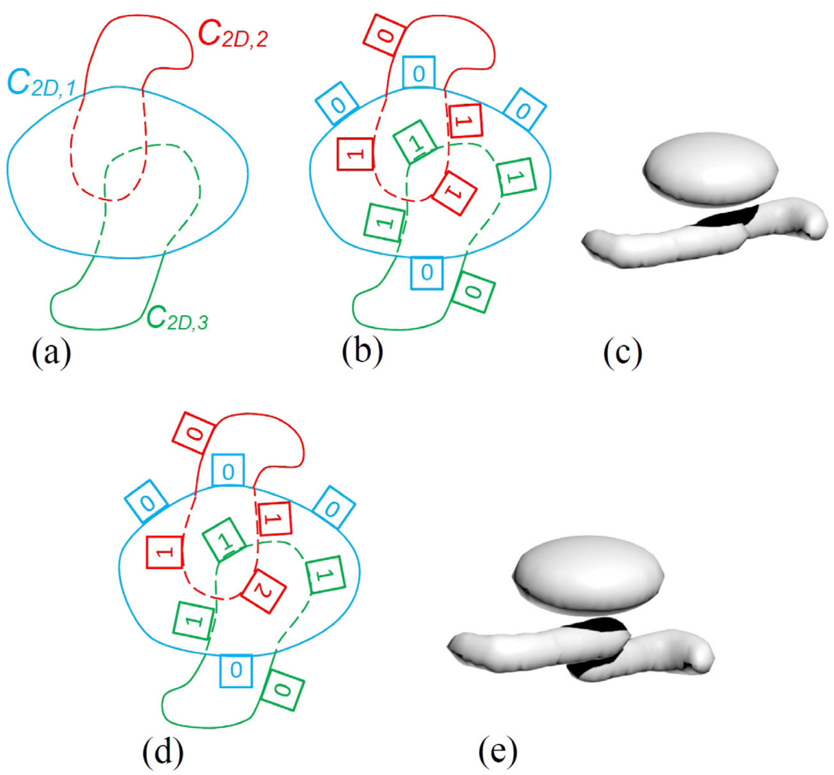


Fig. 3. The input sketch (a); hidden curves have depth index 1 by default (b); the result of the reconstruction shows that the parts of $C_{2D,2}$ and $C_{2D,3}$ intersect since they have a same depth index (c); the depth index assigned by the user to place $C_{2D,2}$ below $C_{2D,3}$ (d); the result of the corresponding reconstruction (e).

depending on the intended shape by the user. By default, both curve segments are assigned with 1 (Fig. 3(b–c)), since the dashed lines encode only one-level of occlusion.

Constraint specification on curve points: Deriving constraints only from the sketch allows little modeling flexibility to the user. In Descriptive, we allow the user to guide the 3D reconstruction of 2D closed curves with the help of specifiers, which are used to define local constraints on one or more selected points. By using corner point specifier, which is user-specified corner (Section 5.2.1), the discontinuities in the z-coordinate of the curve tangent can be defined. The user can also define positional constraints to specify the z-coordinate of some selected curve points (red dots in Fig. 2(e)). Finally, s/he can optionally modify the default value of the planarity coefficient for a selected curve, indicating to which extent the curve is expected to be planar in 3D at the time of reconstruction. For instance, s/he can adjust this coefficient to vary the orientation of the intertwined surface elements in such a way to avoid interpenetration among them (Fig. 7). While these specifiers can be elaborated on 2D curves in the initial viewpoint prior to the initial reconstruction, they can be used after the 3D curve reconstruction in rotated views, allowing an interactive modification of 3D curves or surface meshes from different viewpoints.

Local thickness tuning of the closed surface elements : Once the 2D closed curves have been reconstructed into 3D closed curves, the modeler generates closed surface elements, each corresponding to a closed curve. By default, a surface element has a circular cross-section (Fig. 2(e)). The user has the possibility to adjust the thickness of the surface along z-axis, to create flat shapes (fin in Fig. 2(f)) or concave shapes (cloth in Fig. 19(c)).

4. Surface generation from 3D curves

The system will seek to obtain 3D curves based on the drawing. Once this will be done, the system must generate a closed polygonal mesh based on those curves. The overall procedure is in three steps: (1) we compute an interpolating surface patch for each curve (Fig. 4(b)); (2) we then inflate the surface patches in order to generate volumetric surface elements (Fig. 4(c)); (3) finally, we union the surface elements and perform a smoothing on it to obtain the final shape.

Let $C_{3D,i}$ be a 3D closed curve composed of m points $P_1, P_2, \dots, P_m$ and $C_{2D,i}$ its 2D counterpart on the sketching plane ($z = 0$). We first triangulate the 2D region delimited by $C_{2D,i}$ by using the **Triangle** library by Shewchuk [23] to generate a planar mesh M_{2D}. The triangulation in 2D domain is then mapped to 3D in order to create a 3D surface patch M_{3D}, a lifted version of M_{2D}. From the boundary points of $M_{3D}(C_{3D,i} = P_1, P_2, \dots, P_m)$, we interpolate the z-coordinates of the interior vertices $V = V_1, V_2, \dots, V_o$ by using the mean value coordinates [24] over the 2D domain delimited by $C_{2D,i}$. The z-coordinate $z_{V,j}$ of an interior vertex V_j is defined as follows:

$$z_{V,j} = \sum_{k \in n} (W_k \cdot z_{P,k}) \tag{1}$$

where $z_{P,k}$ is the z coordinate of a curve point P_k, W_k the corresponding weight, and n the number of boundary points used for the mean value coordinates. The result is a surface patch M_{3D} that smoothly interpolates the boundary curve $C_{3D,i}$ (Fig. 4(b)).

Next, the patch M_{3D} is duplicated to generate the front side $M_{3D,fr}$ and the back side $M_{3D,bk}$ of the patch, which are seamed along the boundary. Each of the interior vertices is elevated along the z-axis in order to create a volumetric surface. By default, the z-displacement of each vertex is symmetric with respect to the patch surface and depends on the distance between the vertex and the center of the circumscribed circle of the triangle containing it. The greater the distance, the smaller the displacement (to zero at the edge of the surface). In this way the result is a symmetrical volume element so the cross section is circular.

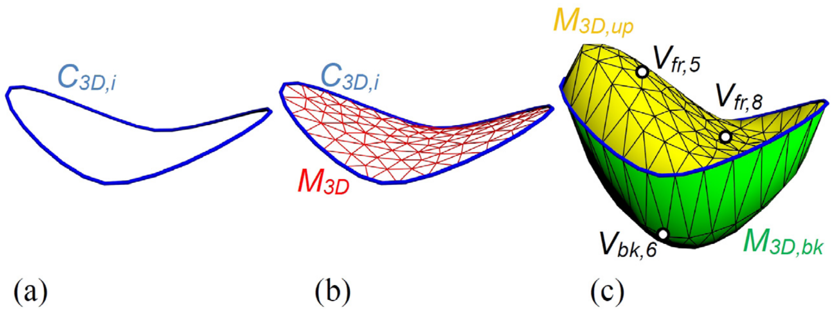


Fig. 4. For a 3D curve $C_{3D,i}$ (a), an interpolating surface patch is generated (b). In (c), the surface patch is duplicated ($M_{3D,fr}$ and $M_{3D,bk}$) and each is inflated to generate a volumetric surface. The thickness coefficients for $V_{fr,5}$, $V_{fr,8}$ and $V_{bk,6}$ have been set to 1.0, −0.03 and 1.0 respectively.

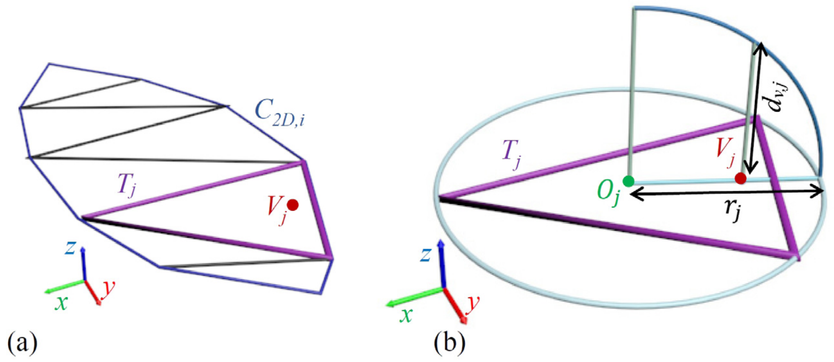


Fig. 5. The triangle T_j that contains V_j is first identified (a). Then, the thickness $d_{V,j}$ at V_j is computed using the center O_j and the radius r_j of the circumcircle, shown in light blue, of T_j. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Let $V_{fr,j}$ and $V_{bk,j}$ be two corresponding vertices one the front and back sides, respectively, which share the same x- and y-coordinates. We derive their z-coordinates as:

$$z_{V,fr,j} = z_{V,j} + \beta_{V,fr,j} \cdot d_{V,j} \qquad (2)$$

$$z_{V,bk,j} = z_{V,j} - \beta_{V,bk,j} \cdot d_{V,j} \qquad (3)$$

- $z_{V,j}$ is the z-coordinate of V_j.
- $d_{V,j}$ denotes the distance to the surface patch (thickness) along z-axis at vertex V_j. To compute this value, the Constrained Delaunay Triangulation (CDT) of $C_{2D,i}$ is first performed in the xy-domain. Next, for each V_j we find its enclosing triangle T_j in the xy-domain (Fig. 5(a)). Let O_j and r_j be the center and the radius of the circumcircle of T_j, respectively (Fig. 5(b)). Then $d_{V,j}$ is defined as $d_{V,j} = \sqrt{r_j^2 - \|O_j - V_j\|^2}$.
- $\beta_{V,fr,j}$ and $\beta_{V,bk,j}$ are coefficients to locally control the thickness of the surface element which are set to 1 by default, while the user can modify them for a set of selected vertices. When the latter happens, the coefficients of all other vertices on the same side are interpolated using the Radial Basis Functions (RBF) [25] with the user specified coefficients as interpolants. Euclidean distance in the xy-domain and Gaussian kernel have been used, with the variance of each kernel set to twice the distance to the nearest neighbor. When there is only one interpolant, the same weight is used for all. Note that the thickness coefficients are determined for each side independently, allowing the creation of a surface with different shapes on each side. The user can generate a concave surface by setting the thickness of one side to a negative value ($V_{fr,8}$ in Fig. 4), or a thin surface by setting both coefficients to 0. Examples of such shapes are shown in Fig. 19(i) and (e), respectively.

Once the meshes of the surface elements have been generated, the final mesh will be obtained by computing the mesh union of all these elements and by applying the Laplacian smoothing [26].

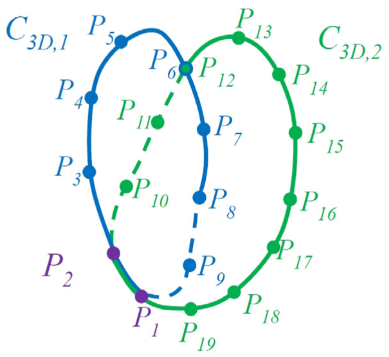
5. 3D reconstruction of 2D closed curves

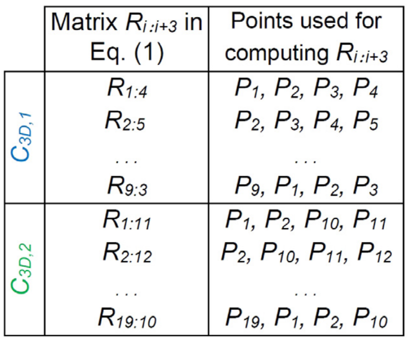
Starting with a set of m simple closed curves $C_{2D} = C_{2D,1}$, $C_{2D,2}$, ..., $C_{2D,m}$ on the sketching plane (z = 0) whose depth ordering has been determined, we proceed to compute their 3D counterparts $C_{3D} = C_{3D,1}, C_{3D,2}, \ldots, C_{3D,m}$ by lifting the curves $C_{2D,j}$ from the sketching plane. This is achieved by solving an optimization problem that finds the optimal 3D curves $C_{3D,j}$ with a set of equality and inequality constraints. In this section we describe the objective function (Section 5.1) and two types of constraints used (Sections 5.2 and 5.3) for the inference of 3D closed curves (Section 5.4).

5.1. Planarity objective function

We compute the 3D closed curves $C_{3D,j}$ that best describe the expected boundary of the intended shape, via optimization. This boils down to the computation of z-coordinate for each point on the 2D closed curves, which had been discretized by using a standard curve sampling algorithm. One criterion that we use is the local planarity, which can be seen as a measure of the smoothness quality of a curve. This means that our modeler regulates the z-coordinates by placing neighboring points in a same plane in 3D. Similar criteria have been adopted in other modelers, such as local affine invariance in [7] or curvature minimization [17].

Let $(p_i, p_{i+1}, p_{i+2}, p_{i+3})$ be 4 consecutive points of $C_{3D,j}$ and (x_i, y_i, z_i), $(x_{i+1}, y_{i+1}, z_{i+1})$, $(x_{i+2}, y_{i+2}, z_{i+2})$ and $(x_{i+3}, y_{i+3}, z_{i+3})$ be their respective coordinates. It follows that the least squares





	Matrix $R_{i:i+3}$ in Eq. (1)	Points used for computing $R_{i:i+3}$
$C_{3D,1}$	$R_{1:4}$	P_1, P_2, P_3, P_4
	$R_{2:5}$	P_2, P_3, P_4, P_5
	...	...
	$R_{9:3}$	P_9, P_1, P_2, P_3
$C_{3D,2}$	$R_{1:11}$	P_1, P_2, P_{10}, P_{11}
	$R_{2:12}$	$P_2, P_{10}, P_{11}, P_{12}$
	...	...
	$R_{19:10}$	P_{19}, P_1, P_2, P_{10}

Fig. 6. Computing the matrices $R_{i:i+3}$ in Eq. (2) for all the points of $C_{3D,1}$ and $C_{3D,2}$ (b). $C_{3D,1}$ and $C_{3D,2}$ have the vertex P_1 and P_7 in common.

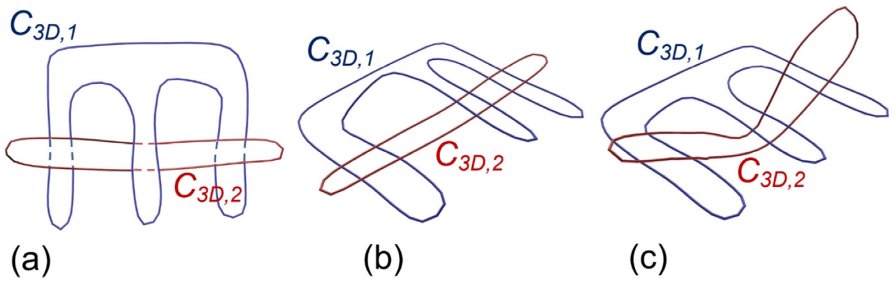


Fig. 7. Using different weight values for the planarity. The curves in the sketching plane (a), the reconstruction with $\alpha_1 = 1$ and $\alpha_2 = 10$ (b), the reconstruction of the same curves with $\alpha_1 = 10$ and $\alpha_2 = 1$ (c).

problem that computes the best fitting plane ($z = a \cdot x + b \cdot y + c$) is:

$$\min_{Q_{i:i+3}} \|P_{i:i+3} \cdot Q_{i:i+3} - Z_{i:i+3}\|^2 \tag{4}$$

where

$$P_{i:i+3} = \begin{pmatrix} x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \\ x_{i+2} & y_{i+2} & 1 \\ x_{i+3} & y_{i+3} & 1 \end{pmatrix}$$

$$Z_{i:i+3} = (Z_i\ Z_{i+1}\ Z_{i+2}\ Z_{i+3})^T$$

$$Q_{i:i+3} = (a_{i:i+3}\ b_{i:i+3}\ c_{i:i+3})^T$$

The unknown variables of the above minimization problem (Eq. (4)) are the plane coefficients $Q_{i:i+3}$. We rewrite this equation so that the unknown variables are the z-coordinates $Z_{i:i+3}$. The solution to Equation (Eq. (4)) is:

$$Q_{i:i+3} = (P^T_{i:i+3} \cdot P_{i:i+3})^{-1} P^T_{i:i+3} \cdot Z_{i:i+3}$$

We compute the vector $Z'_{i:i+3}$ containing the z-coordinates of the 4 points that lie on the solution plane to the above minimization:

$$Z'_{i:i+3} = P_{i:i+3} \cdot Q_{i:i+3} = P_{i:i+3} \cdot ((P^T_{i:i+3} \cdot P_{i:i+3})^{-1} P^T_{i:i+3} \cdot Z_{i:i+3})$$

Finally, we write the minimization problem whose unknown variables are the vector Z_i containing the z-coordinates of the 4 points:

$$\min_{Z_{i:i+3}} \|Z_{i:i+3} - Z'_{i:i+3}\|^2$$

rewritten as :

$$\min_{Z_{i:i+3}} \|R_{i:i+3} \cdot Z_{i:i+3}\|^2 \tag{5}$$

with $R_{i:i+3} = (1 - P_{i:i+3}(P^T_{i:i+3} P_{i:i+3})^{-1} P^T_{i:i+3})$.

Note that the $R_{i:i+3}$ matrix does not involve the plane coefficients; it is a square matrix that depends only on the x- and y-coordinates of the 4 points (P_i, P_{i+1}, P_{i+2}, P_{i+3}).

Note also that $(P^T_{i:i+3} P_{i:i+3})^{-1}$ that is used to compose the $R_{i:i+3}$ matrix may be ill-conditioned when the 4 points are aligned. In this case, we slightly modify the x- and y-coordinates by adding small random noises, i.e. by adding a random value between $[-0{,}0001;0{,}0001]$ to their x- and y-coordinates, so that $R_{i:i+3}$ becomes well-conditioned.

We now define the objective function for the z-coordinates of all points on $C_{3D,j}$ with the planarity energy:

$$\min_{Z} \|R_{C3D,j} \cdot Z\|^2 \tag{6}$$

where Z denotes the vector containing the z-coordinates of the n points of all the curves C_{3D} :

$$Z = (Z_1\ Z_2\ \dots\ Z_n)^T$$

This matrix $R_{C3D,j}$ is filled with the elements of matrices $R_{i:i+3}$ computed for 4 consecutive points starting from point P_i on $C_{3D,j}$. Special care must be taken when two curves intersect or overlap, sharing one or more consecutive points in 2D. When a surface element is partially occluded by another one, their curve counterparts in 2D generally form two separate points (T-junctions) that are shared by the both. In this case, the intersection points are duplicated so that they can have different z-coordinates on each 3D curve (P_6 and P_12 Fig. 6). When two curves share a set of consecutive points, on the other hand, we consider that the corresponding surface elements edges are attached along the shared points thus edges (P_1 and P_2 in Fig. 6). This implies these points share same z-coordinates between two curves.

The objective function is defined for the m curves of C_3D as follows:

$$E_{planarity} = \min_{Z} \|\sum_{j=1}^{m} \alpha_j \cdot R_{C3D,j} \cdot Z\|^2 \tag{7}$$

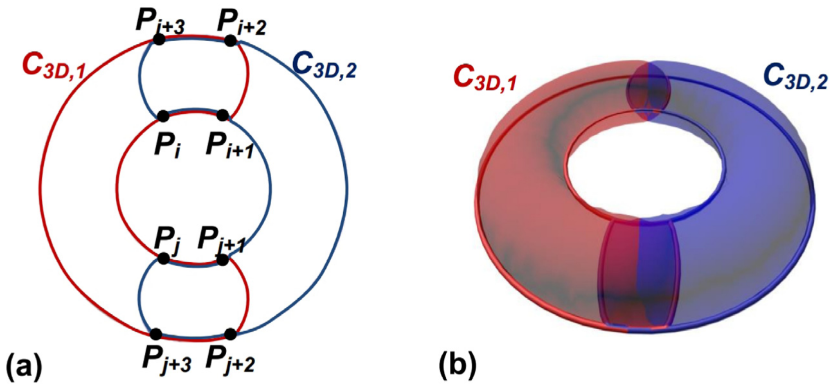


Fig. 8. Modeling of a torus by using two overlapping closed curves $C_{3D,1}$ and $C_{3D,2}$.

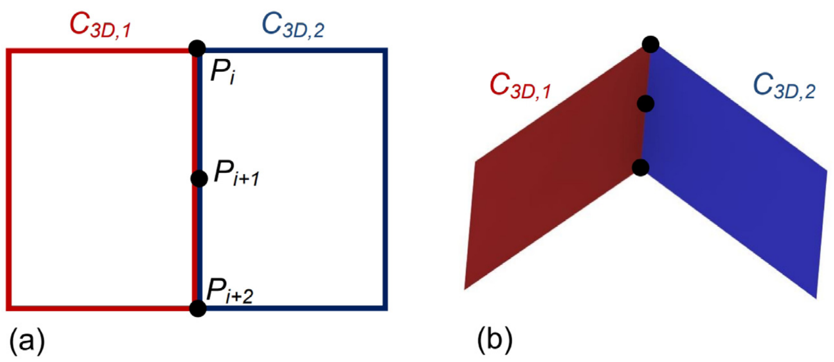


Fig. 9. Modeling of a roof (b) using the two curves in (a). The boundaries of the two surface elements pass through the same points P_i, P_{i+1} and P_{i+2}.

where α_j is the weight of the planarity value of $C_{3D,j}$. The user can modify it to a larger value to make a curve remain planer, or to a smaller one to endow it with more freedom of local reorientation (Fig. 7).

Joining closed shape elements. While in Descriptive the shape is built upon simply closed curves, it is still possible to create shapes with holes by assembling two or more closed curves that partially overlap. Fig. 8 shows the modeling of a torus using two closed curves which pass through the same consecutive vertices P_i, P_{i+1}, P_{i+2}, P_{i+3}, and P_j, P_{j+1}, P_{j+2}, P_{j+3}. Notice that the planarity energy will retain the boundary points of both curves in the plane during the 3D reconstruction. Since they are not aligned, the only way to maintain the planarity is to retain the quadruple points in a same plane, which will subsequently retain their neighboring points along respective boundaries in the same plane. As a result, the two surface elements will stick to each other as they were a single element.

The user can also create surface elements whose boundaries share a same set of points aligned along an edge, forming a ridge, without violating the planarity energy (Fig. 9).

5.2. Local specifiers

In striving to automatically reconstruct a full shape from a single sketch and yet empowering the user the flexibility to design the shape, we provide local specifiers, which are used either to guide or modify the initial 3D reconstruction of curves or surfaces. Below we describe two types of specifiers: positional constraint and corner point.

5.2.1. Corner points specifier

The user can select points on which the planarity rule does not apply, thus the curve abruptly changes its z-coordinate at those selected points. Corner points are useful for creating 3D closed curves that are composed of a set of interconnected planar curves. Fig. 10 shows an example of a 3D curve which is composed of 4 planar curves using 4 corner points. Let P_i be a point of the curve $C_{3D,j}$; the planarity rule involves computing $R_{i-3:i}$, $R_{i-2:i+1}$, $R_{i-1:i+2}$ and $R_{i:i+3}$ (Eq. (5)) corresponding to the following sets of 4 consecutive points starting from P_{i-3}, P_{i-2}, P_{i-1}, and P_i, respectively (Fig. 10(b)). If P_i is selected as a corner point then only $R_{i-3:i}$ and $R_{i:i+3}$ are taken into account for computing $R_{C3D,j}$ in the objective function (Eq. (5)) (Fig. 10(d)). Note that if only one point is present between two corner points, then this point will not be involved in the planarity rule.

5.2.2. Positional constraint specifier

By using a positional constraint specifier, the user can define the z-coordinate of a selected point on the curve. This is used to place a 3D curve at a specifically desired z-coordinate or to generate non-planar 3D curves (Fig. 11).

Positional constraints are written as a set of equalities:

$$\begin{cases} z_i & = & z_{C,i} \\ z_j & = & z_{C,j} \\ \vdots \end{cases}$$

where $(z_i, z_j, \dots)$ denote the z-coordinates of curve points on which the constraints are specified and $(z_{c,i}, z_{c,j}, \dots)$ the user specified values. These equality are written in a matrix form:

$$M_{eq}Z = Z_{eq} \tag{8}$$

where M_{eq} is a square matrix filled with ones (for points with z-coordinate constraints) or zeros (for others) along the diagonal and Z_{eq} denotes the column vector containing the values $(z_{c,i}, z_{c,j}, \dots)$.

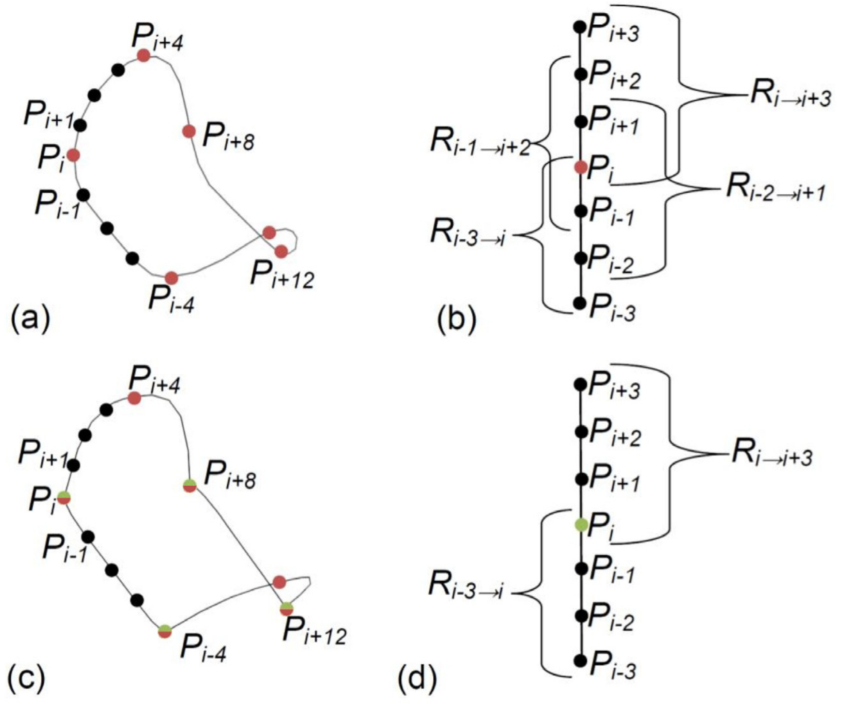


Fig. 10. The $C_{3D,i}$ curve with some positional constraints (a), computing matrices $R_{i-3:i}$, $R_{i-2:i+1}$, $R_{i-1:i+2}$ and $R_{i:i+3}$ for point P_i (b), $C_{3D,i}$ with corner points added to the positional constraints located at P_i, P_{i+8}, P_{i+12} and P_{i-4} (c), computing matrices $R_{i-3:i}$ and $R_{i:i+3}$ for point P_i(d).

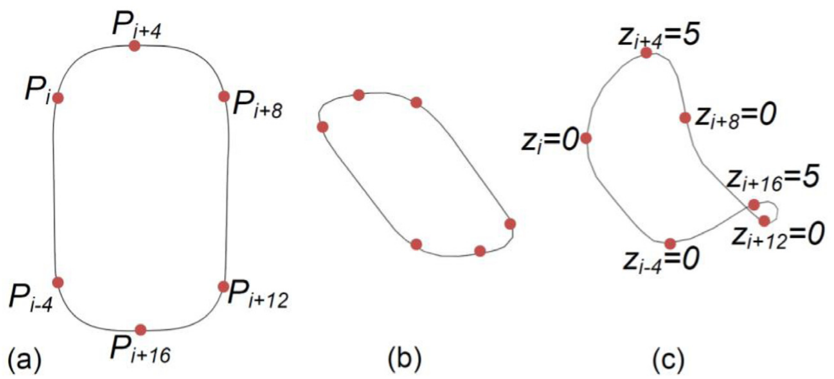


Fig. 11. The $C_{2D,i}$ curve (a), the corresponding $C_{3D,i}$ curve computed using only the planarity property (b), $C_{3D,i}$ with positional constraints specified at P_i, P_{i+4}, P_{i+8}, P_{i+12}, P_{i+16} and P_{i-4}.

5.3. Depth ordering of 2D curve segments

For each 2D curve segment C, we define a depth index $I(C) \in \mathbb{D} = \{0, 0.5, 1, 1.5, 2, \ldots\}$. $I(C_i) < I(C_j)$ if the region enclosed by C_i is partially or entirely occluding that of C_j in the sketching view. These depth indices are assembled as inequality constraints into the optimization problem to ensure that the relative depth order between 3D curves is consistent with the input drawing.

By default, the curve depth indices are set to 0 or 1, depending on the visibility of the curve on the input drawing. These values are updated by analyzing their corresponding 2D curve junctions that are formed at their intersection points on the sketching plane. In what follows, we explain the rules for determining the depth indices of the segments by using the cusps and three types of junctions: T-junctions, X-junction and Y-junctions. Note that only junctions between two curves are considered. Our system does not handle junctions of three or more curves.

5.3.1. T-junction

A T-junction corresponds to a crossing, an intersection, between two curves, at a single point. These junctions always go in pairs, because they appear when two closed curves overlap, and correspond to the points where one of these two regions passes under the other (Fig. 12(a)). When a pair is detected the system looks if at least one of the two junctions corresponds to a point where a curve goes from visible (index of 0) to hidden (index of 1 or more) in order to determine the depth order between these two elements. If we define the two curves as $C_{2D,i}$ (the element above) and $C_{2D,j}$ (the element below) and their two intersection points $P_{T,1}$ and $P_{T,2}$, with $P_{T,1}$ corresponding to the vertex where the change of visibility of $C_{2D,j}$ takes place, then the system assigns to the part of the curve $C_{2D,j}$ delimited by $[P_{T,1}, P_{T,2}]$ a depth index equal to $n + 1$ with n corresponding to the maximum depth index of curve $C_{2D,i}$ at the two junctions. This is useful when multiple elements overlap (Fig. 12(b)).

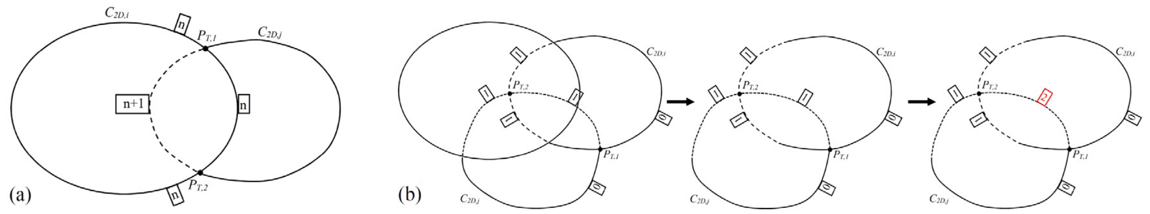


Fig. 12. Method to determine depth indices when two curves overlap (a). Example where this rule allows to deduce the order between two elements belonging to an overlap of 3 elements (b). Initially the solid lines have an index of 0 and the dashed line of 1. Once the program detects the pair of T-junctions between the two elements $C_{2D,i}$ and $C_{2D,j}$, it updates the value of the depth indices of the segment $C_{2D,j}$ delimited by $[P_{T,1}, P_{T,2}]$ as the maximum value of the indices of $C_{2D,i}$ at these junctions plus 1, so in this case at 2.

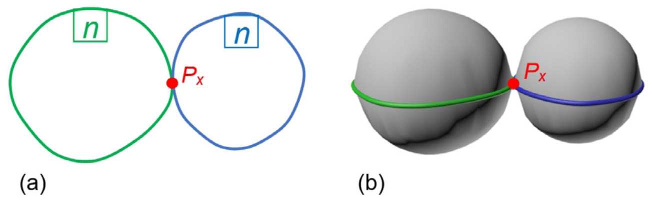


Fig. 13. Two close curves touching each other at the X-junction P_X (a). The reconstructed surfaces are attached to each other at P_X (b).

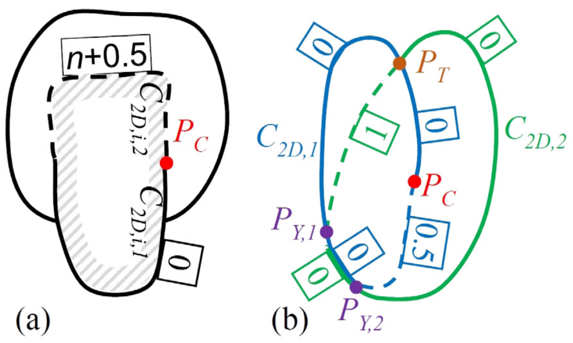


Fig. 14. Computation of depth indices of the curve segments connected at a cusp P_C (a); the occluded curve segment is shown in dashed line. Example of a drawing composed of two curves $C_{2D,1}$ and $C_{2D,2}$ with a T-junction P_T, a cusp P_C and two Y-junctions $P_{Y,1}$ and $P_{Y,2}$ (b).

5.3.2. X-junction

X-junctions are points where two close curves touch each other (see Fig. 13(a)) but do not cross each other unlike T-junctions; these two curves have only one point in common, which is the points where the two curves are tangent to each other. These two curves are located at the same depth in 3D (see Fig. 13(b)) and therefore, are assigned with the same depth index *n*.

5.3.3. Y-junction

A Y-junction is a point where two curves overlap, i.e. pass through a same set of consecutive points. Y-junctions are present in two different forms: Y-junctions with no occluded segment and Y-junctions with one occluded segment. An example of a Y-junctions with occluded segment is present at Fig. 14(b). Unlike cusps and T-junctions, Y-junctions do not allow determining the depth indices.

5.3.4. Cusp junction

Cusps are the endpoints of visible curves. Unlike T-junctions, however, cusps are not connected to any other curve. A cusp separates a visible curve segment from a hidden segment; it is a point P_C where the curve $C_{3D,i}$ goes inside the volume delimited by a surface element.

If we define n as the depth index of $C_{2D,i,1}$, then an index of $n + 0.5$ is assigned to its hidden part. An index with a decimal portion of 0.5 indicates that the 2D curve lies within another element in 3D, that is to say between its front and its back face.

5.3.5. Expression of inequality constraints

In order to compute the depth indices of the curve segments, we adopt an approach similar to [27]. The T-junctions, X-junctions, and cusps determine how the depth indices of the curve segments change at the junctions. Y-junctions are not considered since they do not allow determining the depth indices. For each junction, an integer equality that relates the depth index of the curve segments connected to this junction is written. All the curve segments which are not hidden are assigned with depth index 0. The depth indices of all hidden curve segments are then computed by solving the integer linear program consisting of all the integer equalities. In case the linear program does not have a solution, the user is asked to modify the curves and their junctions. Once the depth indices are computed, a set of inequality constraints are written for the curve segments. The purpose of these inequalities is to keep a minimum distance along the z-axis between the curves of the surface elements that are overlapping in the sketching (x,y) plane.

Let $C_{3D,i,j}$ be a curve segment and $d_{i,j}$ its depth index. If $d_{i,j}$ is a round number, it implies that the segment is located between two surface elements along the z-axis. Thus, we identify E_B and E_F which are the elements behind and in front of $C_{3D,i,j}$ respectively by looking at the neighboring segments of $C_{3D,i,j}$. We then write a set of inequalities for $C_{3D,i,j}$ and E_B to keep the z-coordinates of $C_{3D,i,j}$ larger than those of E_B:

$$M_{juncEB} \cdot Z > D_{juncEB} \qquad (9)$$

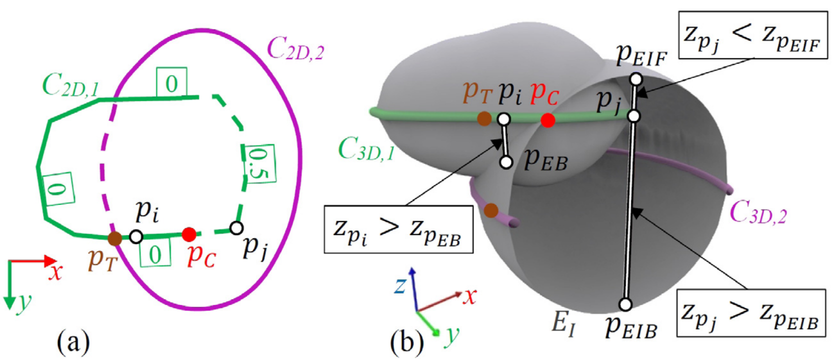


Fig. 15. The curves $C_{2D,1}$ and $C_{2D,2}$ with a T-junction p_T and a cusp p_C (a). The constraint inequalities for the points p_i and p_j of $C_{3D,1}$ (b).

Z is the column vector of size N containing the z-coordinates of all the points of the curves C_{3D}. Each line of M_{juncEB} and D_{juncEB} corresponds to an inequality constraint:

$$z_{p_K} > z_{p_{EB}}$$

where z_{p_k} is the z-coordinates of a point p_k of the segment $C_{3D,i,j}$ and z_{p_EB} is z-coordinate of a point p_{E_B} in the front side of the element E_B. p_k and p_{E_B} have same x and y-coordinates and the z-coordinate of p_{E_B} is expressed as a weighted sum of the z-coordinates of the points of the curve of E_B (see Eqs. (1), (2) and (3) in Section 4). Similarly, a set of inequalities is also written for E_F and $C_{3D,i,j}$:

$$M_{juncEF} \cdot Z < D_{juncEF} \tag{10}$$

Each line of this set is an inequality between the z-coordinate of a point p_l of $C_{3D,i,j}$ and the z-coordinate of the corresponding point p_{E_F} on the back side of the element E_F. Note that the sign of the inequality is inverted since the surface element E_F is in front of $C_{3D,i,j}$ and not behind.

A depth index $d_{i,j}$ of the segment $C_{3D,i,j}$ with a decimal part of 0.5 indicates that $C_{3D,i,j}$ is located inside the volume of a surface element E_I (i.e. the case of cusp). This element E_I is found by looking at the neighboring segments of $C_{3D,i,j}$ and their depth indices. We then write a set of inequalities to keep the z-coordinates of the points of $C_{3D,i,j}$ smaller than the z-coordinates of the points in the front side of E_I:

$$M_{juncEIF} \cdot Z < D_{juncEIF} \tag{11}$$

Each line of this set of equations is an inequality of the form:

$$Z_{p_l} < Z_{p_{EIF}}$$

where z_{p_l} is the z-coordinate of a point p_l of the segment $C_{3D,i,j}$ and p_{EIF} is the point with the same x and y-coordinate in the front surface of E_I.

Similarly, we write a set of inequalities to keep the z-coordinates of the points of $C_{3D,i,j}$ larger than the z-coordinates of the points in the back side of E_I:

$$M_{juncEIB} \cdot Z > D_{juncEIB} \tag{12}$$

Each line of this set is an inequality of the form:

$$z_{p_l} > z_{p_{EIB}}$$

where p_{EIB} is the point with the same x and y-coordinate in the surface E_{IB}.

In this way, each point p_l of the segment $C_{3D,i,j}$ will be subject to two inequalities, one placing it in front of the back face of E_I, the other placing it behind the front face of E_I. Therefore the curve $C_{3D,i,j}$ is placed inside E_I. Fig. 15 shows the case of a cusped junction and the inequalities that are deduced from it.

5.4. Solving the optimization problem

The z-coordinates of curve points are found by solving the following quadratic optimization problem, where the planarity energy is minimized subject to two sets of constraints:

$$\min_Z \| \sum_{j=1}^{m} \alpha_j \cdot R_{C3D,j} \cdot Z \|^2 \ s.t. \begin{cases} M_{eq} \cdot Z & = & Z_{eq} \\ M_{ineq} \cdot Z & < & D_{ineq} \end{cases}$$

where Z is the vector containing the z-coordinates of all points of C_{3D}:

$$Z = (z_1 \ z_2 \ \dots \ z_n)^T$$

The equalities $M_{eq} \cdot Z = Z_{eq}$ are from Eq. (8). The inequalities $M_{ineq} \cdot Z < D_{ineq}$ are composed of the inequalities of Eqs. (9)–(12). Note that the linear inequalities may not have any solution. This happens when the surface elements are so tightly interlaced that the inequality constraints for the T-junctions cannot be satisfied all together. Note also that this optimization may have an infinite number of solutions. This occurs when no positional constraints are defined; the optimization does not include any equality constraint.

The optimization problem is solved with a linear least-squares solver. Among the many available algorithms [28], we chose the interior point algorithm [29], since it was the most efficient and terminated with a maximum of 2000 iterations. Another advantage of this algorithm is its ability to compute a solution even for the case when the optimization has an infinite number of solutions.

The solution of this optimization is a set of 3D curves, which are presented in a 3D viewer. The user can add new positional constraints or new corner points in order to refine the result by computing a successive optimization. This process can be repeated until the desired 3D shape is obtained.

6. Results and discussion

Descriptive has been implemented in C++ as a plugin for 3DS Max running on Windows 10. Experiments were performed on a workstation with 16 GB of memory and an Intel Core i7-6700 processor running at 3.4 GHz. In Fig. 19 we show more of our results. The constraints specified by the user are shown in red for positional constraints, green for corner points, and green–red for both. As we can see, a large variety of shapes can be modeled, including concave shapes such as a vase (Fig. 19(f)), a pipe (Fig. 19(g)) or layered shapes such as a clothed teddy bear (Fig. 19(c)). As well, fully polyhedral objects like a house (Fig. 19(i)) have been constructed. Notice how well our method manages occluding parts and can seamlessly reconstruct shapes even with self-occlusion (Fig. 19(b), (d)). Notice also that a failure

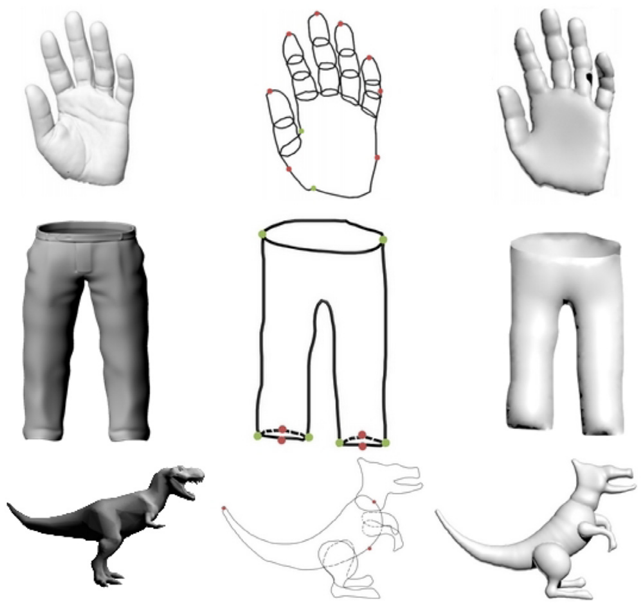


Fig. 16. Using a training example ('hand') composed of a 3D model and its corresponding sketch, the user learns how to decompose the sketch and uses local specifiers to obtain 3D shape. The other two rows ('pants' and 'dinosaurs') show test examples. From left to right: the target models that the user was given as references, the sketch provided by the user with the corner point (green) and positional constraints (red), and the reconstructed models. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

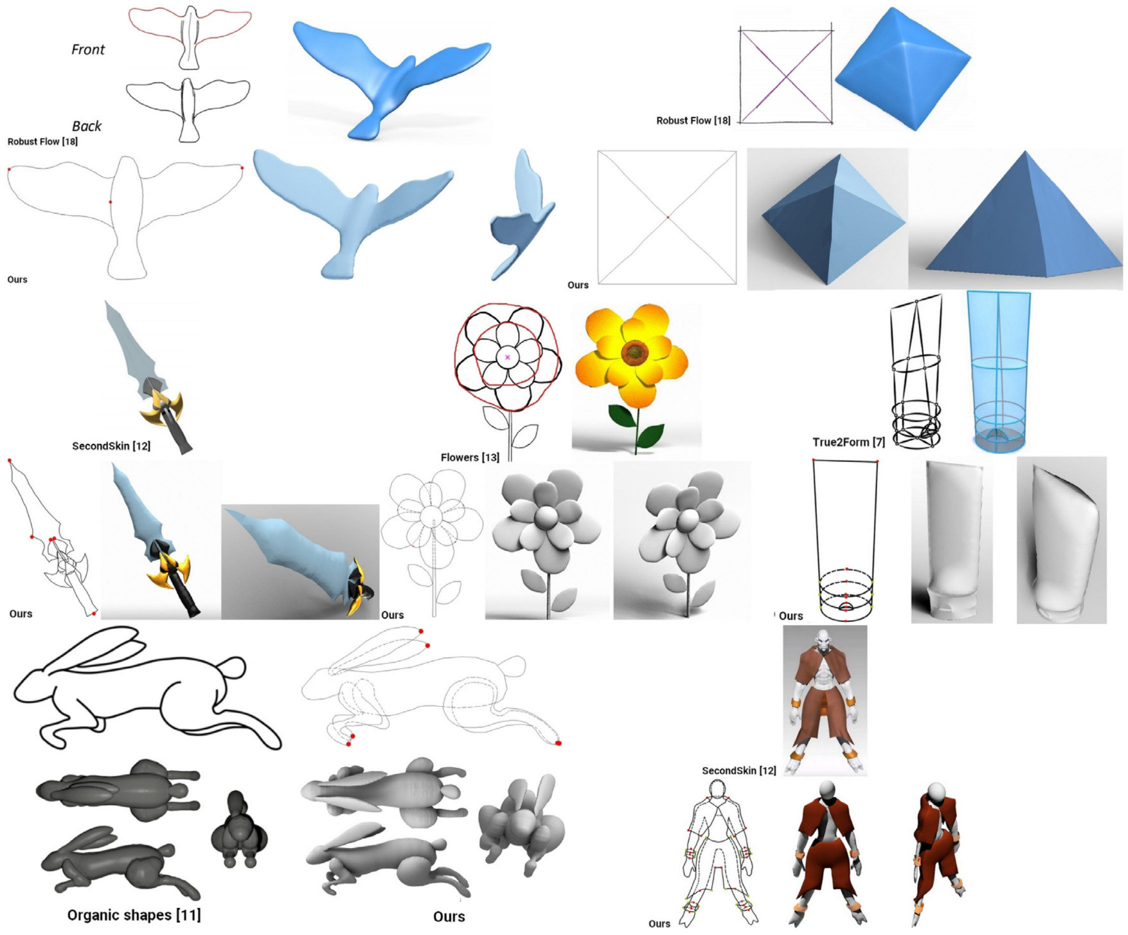


Fig. 17. Visual comparison with state-of-the-art methods. Each result shown in the figure is accompanied with the input sketch used by each method whenever available in their article.

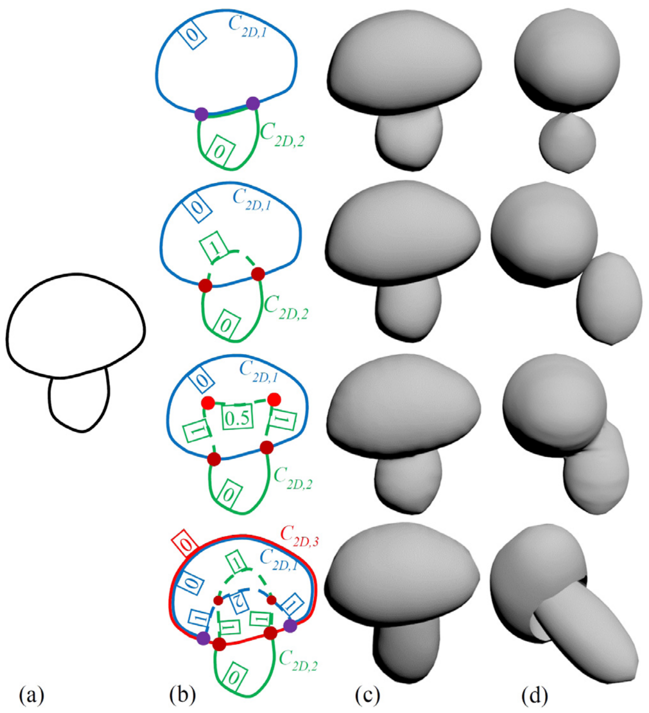


Fig. 18. The same sketch in (a) can result in different shapes depending on the way how the user defines the hidden curves and their depth index (b). T-junctions, cusps and Y-junctions are shown in brown, red and purple respectively. The results of the reconstruction are shown in front view (c) and perspective view (d). The decomposition may not always be straightforward and may require some insights on the modeler. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

case for Entem et al. [11] has been easily constructed by our modeler (Fig. 19(e)). In addition, our method does not imply any specific viewpoint and can reconstruct a model with possibly different orientations for each part of an object, such as the kangaroo in Fig. 19(j). The time required to produce these results varied from 5 min (propeller Fig. 19(e)) to 25 min (tangled tree in Fig. 19(d)), including the sketching time. On average it took 7 s to detect the junctions, assign the depth indexes and compose an optimization problem with the constraints, and 20 s to solve it. Note that the computation time greatly depends on the complexity of the shape (usually the numbers of closed elements and junctions) present in the sketch. For the tangled tree (Fig. 19(d)) those times are respectively 11 and 41 s for an optimization problem with a vector Z of 1073 variables and subjected to 766 constraints, whereas for the propeller (Fig. 19(e)) they are almost instantaneous (1 and 2 s, respectively, for 103 variables and 25 constraints).

Note that the number of constraints necessary to the reconstruction of each model is minimal. For the dinosaur for example, only 3 positional constraints are necessary, placed on arbitrary points and with $Z = 0$ with the sole objective of placing the reconstructed model on the plane $Z = 0$. This result is therefore obtained directly from the single view of the drawing, without having to go through rotations of other 3D views. For the other models, the positional constraints are mainly used after the first reconstruction to specify a specific position, among all those possible, corresponding to what the artist wants. For the chair (b) for example the user specifies the position of the top and bottom of each chair leg to ensure that they are all perfectly vertical. We provide the full set of results in the supplemental materials.

6.1. User evaluation

We invited 8 users to take part in our user evaluation. At the first stage each user learned how to use Descriptive (20 min in average). From three example models and their corresponding sketches, they first learned how to decompose the sketch into closed curves, and to use local specifiers. Next, they were asked to sketch by referring to given 3D models and to reconstruct in 3D by following the full pipeline, which took 6 9 min. Fig. 16 shows one of the training examples and two results produced by the users. Finally, we asked the participants to rate their experience and our modeler from 1 (very negative) to 5 (very positive). On average users gave a score of 4.6 to the ease of drawing descriptively and a score of 4.1 to the overall ease of use of our modeler. They also rated 4.75 on how their 3D results corresponded to their expectations.

Two participants among the 8 had previous experience with other 3D modeling packages and were asked to compare our modeler with those packages. They both liked the way they could obtain a 3D model from a 2D drawing quite easily and within a short time of learning, compared to the 3D modelers they had used. They also found that our modeler shortened the time for creating 3D shapes.

6.2. Comparison with previous work

6.2.1. Visual comparison

Fig. 17 compares some of our results with models obtained in other related work. It is important to point out that other methods do not take as input a single descriptive sketch as ours. Therefore, each result shown in the figure is accompanied with the input sketch used by each method whenever available in their article. To compare with 'Secondskin [12]', we have drawn input sketches to our modeler by referring to their result models. Overall our method manages to produce results with comparable quality, and even better in some cases, than those from other works, while requiring on average less sketch input. For example, our method requires only a single sketch for the bird compared to the other ('Robust flow' [18]) requiring two. The curve input

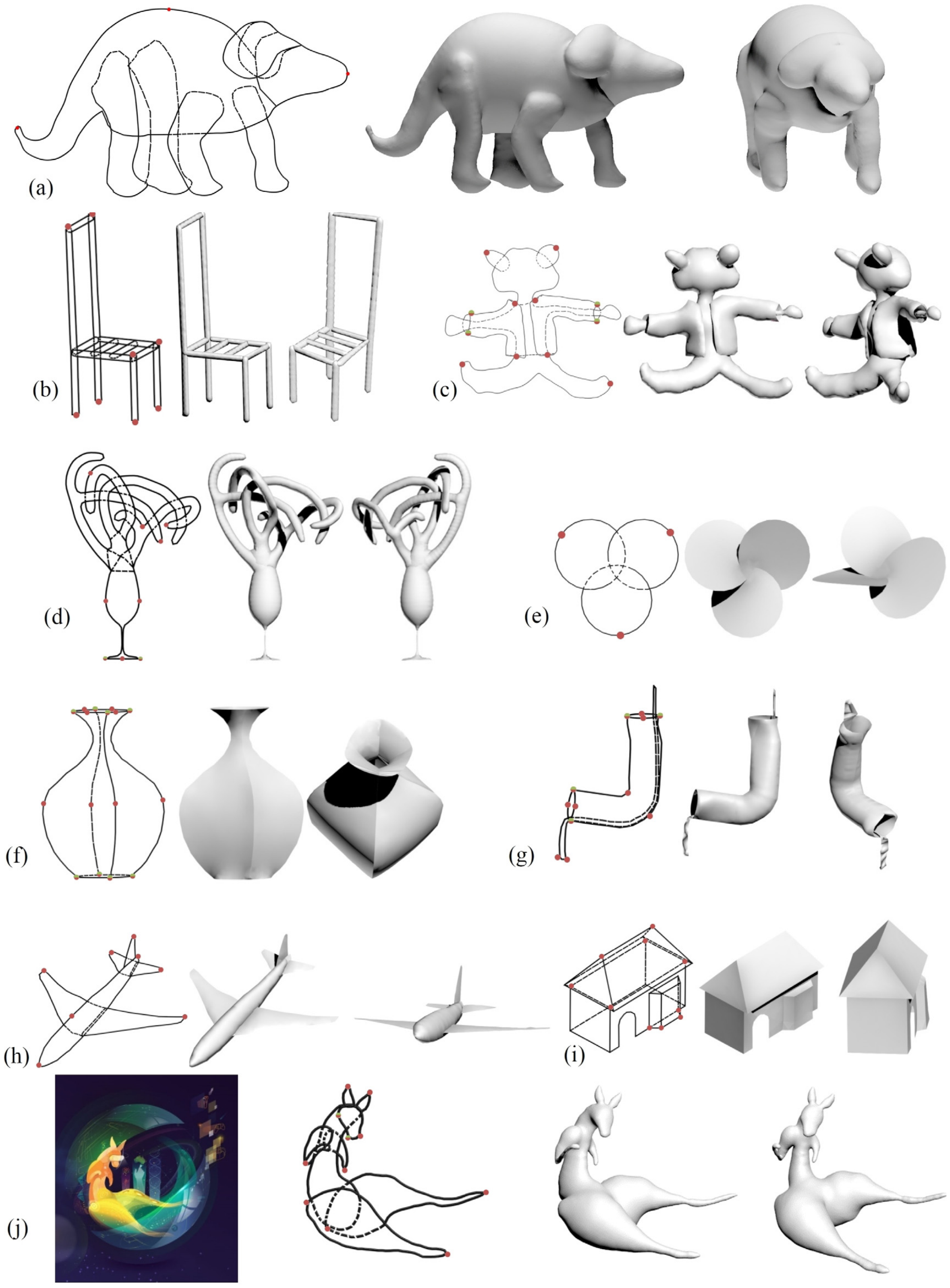


Fig. 19. Results obtained with our modeler. The constraints specified by the user are shown in red for positional constraints, green for corner points, and green–red for both. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

to create the toothpaste's tube is lighter in our modeler than in 'True2Form' [7].

In the Rabbit's example produced by the fully automatic method for organic shape [11], the small drawn parts of the front and rear legs that are greatly occluded by other parts of the body are simply ignored and are generated as symmetric counterpart of the fully drawn legs, thus do not correspond to the input sketch. Subsequently, even though the 3D result is plausible, it does not totally correspond to the user-intended pose. The flaw of their result is also visible at the ears of the rabbit which are badly placed on the head and overlap each other. The use of the descriptive style in our method allows us to overcome all these imperfections. The legs are fully drawn so the 3D result corresponds perfectly to the curves provided by the user. The ears of our result are also well separated from each other, as they are fully drawn (and thus, the ambiguity of the position on the head of the partially hidden ears is non-existent). Moreover our modeler assures that each of those ears is placed correctly in 3D, since the user can specify the position of the tips of each ear.

6.2.2. Discussion

We discuss now our results in a comparative manner with respect to the state-of-the-art methods. Fig. 19 demonstrates the reconstruction of organic shapes (dinosaur: Fig. 19(a), kangaroo: Fig. 19(j)), manufactured objects (chair: Fig. 19(b), vase: Fig. 19(f), pipe: Fig. 19(g), airplane (Fig. 19(h)), objects with self-occlusion (chair: Fig. 19(b), tree: Fig. 19(d)), with surface patch (house: Fig. 19(i), propeller: Fig. 19(e), vase: Fig. 19(f)), and with layers (teddy bear: Fig. 19(c), pipe: Fig. 19(g)).

Most of existing methods are designed to work well for a specific type of shapes, and thus can reconstruct only organic shapes [11], or only manufactured objects [7], but not both.

Methods that can model general shapes usually require to work with multi-view sketch, and do not allow the same variety of shapes as our modeler. Teddy [30] can reconstruct organic shapes as well as manufactured objects, but cannot produce results with surface patches. It is also difficult to work with sketches containing self-occluding elements or layered ones. Similarly, the learning-based modeler [18] can generate shapes that can be decomposed into a limited number of occlusion-free surface patches, but will not treat well those with self-occlusion, or layered objects. BendSketch [6] can reproduce most of our results, but with some difficulties. For example, shapes with layers or with many surface patches require considerable effort to the user as s/he has to separately work on different components and then merge them together. Moreover, it cannot handle self-occluding objects.

Compared to already existing methods, Descriptive allows the reconstruction of a real 3D mesh (not a relief such as [1–3]) without being limited on the nature of the object (man-made or organic shape) or to objects with a particular geometry (such as symmetry [31]). By allowing the user to easily change the position of certain elements through position constraints, the reconstructed models will not be limited in depth and will not give that impression of flattened shape that can be obtained with other methods such as [17] and [32]. Reconstruct the model of the chair with these methods would be impossible for example. Moreover, to our knowledge, very few SBMs, including those that incrementally reconstruct very complex shapes such as ShapeShop [33], are able to reconstruct an element inside another encompassing element like the cylinder in which flows water or cloth around teddybear. We can reconstruct each of the models we have in our results by using specifically for each one an existing suitable SBM, but Descriptive is able to reconstruct itself the integrality of these results, and all this by requiring only a learning time of about 20 min as shown in our user study.

6.3. Limitations

Although we provide a method for quick and easy reconstruction of a wide variety of 3D models, some room for improvement still exists. Some 3D models can sometimes require a considerable amount of time and effort for the placement of geometric constraints; in particular, this is the case for shapes whose 3D silhouette curves have a lot of intertwining, like a complex interlacing of pipe for example. Moreover, the drawing of the hidden parts is not always obvious for a non-experienced user. As shown in Fig. 18, a user may find herself/himself in a situation where several seemingly correct 2D sketches would result in completely different 3D models. In order to obtain an expected model, the user needs to have some insights on the modeler and differentiate between these similar inputs.

Moreover, our method is more appropriate for the modeling of the rough shape of an object. Adding fine surface details with our approach can be very difficult, if not impossible. A good way of compensating for this limitation would be to combine our method with Bendsketch [6]

7. Conclusion

In this paper we have presented Descriptive, a sketch-based modeler that reconstructs 3D shapes from a single 2D descriptive sketch. By combining the representation power of descriptive sketch and the optional user specification, it extends the range and complexity of shapes that can be created from a single-view sketch, while maintaining the intuitive, labor-efficient sketching interface. Compared to alternative sketch-based modelers, there is no need to parse different kinds of sketch strokes and the user is free to choose an arbitrary view. Together with the user specified constraints, the 3D reconstruction problem is solved as a linear optimization problem with equality and inequality constraints, thus saving much computation time.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research has been supported by the French Ministry of Higher Education, Research, and Innovation.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cad.2020.102904.

References

[1] Sýkora D, Kavan L, Čadík M, Jamriška O, Jacobson A, Whited B, et al. Ink-and-Ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters. ACM Trans Graph 2014;33(2):16.

[2] Yeh C, Huang S, Jayaraman PK, Fu C, Lee T. Interactive high-relief reconstruction for organic and double-sided objects from a photo. IEEE Trans Vis Comput Graphics 2017;23(7):1796–808. http://dx.doi.org/10.1109/TVCG.2016.2574705.

[3] Dvorožňák M, Nejad SS, Jamriška O, Jacobson A, Kavan L, Sýkora D. Seamless reconstruction of part-based high-relief models from hand-drawn images. In: Proceedings of international symposium on sketch-based interfaces and modeling. 2018.

[4] Nealen A, Igarashi T, Sorkine-Hornung O, Alexa M. Fibermesh: designing freeform surfaces with 3d curves. ACM Trans Graph 2007;26:41.

[5] Wang K, Zheng J, Seah HS. Progressive sketching with instant previewing. Comput Graph 2019;81:9–19. http://dx.doi.org/10.1016/j.cag.2019.03.017, URL http://www.sciencedirect.com/science/article/pii/S0097849319300408.

[6] Li C, Pan H, Liu Y, Sheffer A, Wang W. Bendsketch: Modeling freeform surfaces through 2d sketching. ACM Trans Graph (SIGGRAPH) 2017;36(4):125:1–125:14. http://dx.doi.org/10.1145/3072959.3073632.

[7] Xu B, Chang W, Sheffer A, Bousseau A, McCrae J, Singh K. True2form: 3D curve networks from 2d sketches via selective regularization. Trans Graph (Proc. SIGGRAPH 2014) 2014;33(4). http://dx.doi.org/10.1145/2601097.2601128.

[8] Iarussi E, Bommes D, Bousseau A. Bendfields: Regularized curvature fields from rough concept sketches. ACM Trans Graph 2015. URL http://www-sop.inria.fr/reves/Basilic/2015/IBB15.

[9] Nishida G, Garcia-Dorado I, Aliaga DG, Benes B, Bousseau A. Interactive sketching of urban procedural models. ACM Trans Graph 2016;35(4):130:1–130:11. http://dx.doi.org/10.1145/2897824.2925951, URL http://doi.acm.org/10.1145/2897824.2925951.

[10] Schwärzler M, Kellner L-M, Maierhofer S, Wimmer M. Sketch-based guided modeling of 3D buildings from oriented photos. In: Proceedings of the 21st ACM SIGGRAPH symposium on interactive 3d graphics and games. I3D '17, New York, NY, USA: ACM; 2017, p. 9:1–8. http://dx.doi.org/10.1145/3023368.3023374, URL http://doi.acm.org/10.1145/3023368.3023374.

[11] Entem E, Parakkat Ad, Barthe L, Muthuganapathy R, Cani M-P. Automatic structuring of organic shapes from a single drawing. Comput Graph 2019. URL https://hal.inria.fr/hal-02058765.

[12] Paoli CD, Singh K. Secondskin: sketch-based construction of layered 3D models.. ACM Trans Graph 2015;34(4):126:1–126:10, URL http://dblp.uni-trier.de/db/journals/tog/tog34.html#PaoliS15.

[13] Bobenrieth C, Seo H, Cordier F, Habibi A. Reconstructing flowers from sketches. In: Fu H, Ghosh A, Kopf J, editors. Comput Graph Forum 2018;37(7):167–78. http://dx.doi.org/10.1111/cgf.13557.

[14] Huang H, Kalogerakis E, Yumer E, Mech R. Shape synthesis from sketches via procedural models and convolutional networks. IEEE Trans Vis Comput Graphics 2017;23(8):2003–13. http://dx.doi.org/10.1109/TVCG.2016.2597830.

[15] Bessmeltsev M, Chang W, Vining N, Sheffer A, Singh K. Modeling character canvases from cartoon drawings. Trans Graph (2015) 2015;34(5). http://dx.doi.org/10.1145/2801134.

[16] Bessmeltsev M, Vining N, Sheffer A. Gesture3d: posing 3d characters via gesture drawings. ACM Trans Graph 2016;35:165:1–165:13.

[17] Cordier F, Seo H. Free-form sketching of self-occluding objects. IEEE Comput Graph Appl 2007;27(1):50–9. http://dx.doi.org/10.1109/MCG.2007.8.

[18] Li C, Pan H, Liu Y, Sheffer A, Wang W. Robust flow-guided neural prediction for sketch-based freeform surface modeling. ACM Trans Graph (SIGGRAPH ASIA) 2018;37(6):238:1–238:12. http://dx.doi.org/10.1145/3272127.3275051.

[19] Han X, Gao C, Yu Y. Deepsketch2face: A deep learning based sketching system for 3d face and caricature modeling. 2017, CoRR arXiv:abs/1706.02042. arXiv:1706.02042. URL http://arxiv.org/abs/1706.02042.

[20] Delanoy J, Aubry M, Isola P, Efros A, Bousseau A. 3d sketching using multi-view deep volumetric prediction. Proc ACM Comput Graph Interact Tech 2018;1(21). URL http://www-sop.inria.fr/reves/Basilic/2018/DAIEB18.

[21] Lun Z, Gadelha M, Kalogerakis E, Maji S, Wang R. 3d shape reconstruction from sketches via multi-view convolutional networks. In: 2017 international conference on 3d vision (3DV). 2017.

[22] Pusch R, Samavati F, Nasri A, Wyvill B. Improving the sketch-based interface: Forming curves from many small strokes. Vis Comput 2007;23(9–11):955–62. http://dx.doi.org/10.1007/s00371-007-0160-5, URL http://link.springer.com/10.1007/s00371-007-0160-5.

[23] Shewchuk JR. Triangle: Engineering a 2D quality mesh generator and delaunay triangulator. In: Lin MC, Manocha D, editors. Applied computational geometry: Towards geometric engineering. Lecture notes in computer science, vol. 1148, Springer-Verlag; 1996, p. 203–22, From the First ACM Workshop on Applied Computational Geometry.

[24] Hormann K, Floater MS. Mean value coordinates for arbitrary planar polygons. ACM Trans Graph 2006;25(4):1424–41. http://dx.doi.org/10.1145/1183287.1183295, URL http://doi.acm.org/10.1145/1183287.1183295.

[25] Buhmann MD. Radial basis functions: Theory and implementations. Cambridge monographs on applied and computational mathematics, Cambridge University Press; 2003, http://dx.doi.org/10.1017/CBO9780511543241.

[26] Vollmer J, Mencl R, Muller H. Improved Laplacian smoothing of noisy surface meshes. Comput Graph Forum 1999. http://dx.doi.org/10.1111/1467-8659.00334.

[27] Williams LR. Topological reconstruction of a smooth manifold-solid from its occluding contour. In: Eklundh J-O, editor. Computer vision — ECCV '94. Berlin, Heidelberg: Springer Berlin Heidelberg; 1994, p. 36–47.

[28] Nocedal J, Wright SJ. Numerical optimization, 2nd ed.. New York, NY, USA: Springer; 2006.

[29] Gondzio J. Multiple centrality corrections in a primal–dual method for linear programming. Comput Optim Appl 1995;6:137. http://dx.doi.org/10.1007/BF00249643.

[30] Igarashi T, Igarashi T, Matsuoka S, Tanaka H. Teddy: A sketching interface for 3d freeform design. In: ACM SIGGRAPH 2007 courses. SIGGRAPH '07, New York, NY, USA: ACM; 2007, http://dx.doi.org/10.1145/1281500.1281532, URL http://doi.acm.org/10.1145/1281500.1281532.

[31] Cordier F, Seo H, Park J, yong Noh J. Sketching of mirror-symmetric shapes. IEEE Trans Vis Comput Graphics 2011;17:1650–62.

[32] Karpenko OA, Hughes JF. Smoothsketch: 3D free-form shapes from complex sketches. ACM Trans Graph 2006;25:589–98.

[33] Schmidt R, Wyvill B, Sousa MC, Jorge JA. Shapeshop: Sketch-based solid modeling with blobtrees. In: ACM SIGGRAPH 2007 courses. SIGGRAPH '07, New York, NY, USA: ACM; 2007, http://dx.doi.org/10.1145/1281500.1281554, URL http://doi.acm.org/10.1145/1281500.1281554.