Pacific Graphics 2017
J. Barbic and O. Sorkine-Hornung
(Guest Editors)

Volume 36 (2017), Number 7

Semi-Automatic Conversion of 3D Shape into Flat-Foldable Polygonal Model

Emi Miyamoto Yuki Endo Yoshihiro Kanamori Jun Mitani

University of Tsukuba

Abstract

This paper presents a method that can convert a given 3D mesh into a flat-foldable model consisting of rigid panels. A previous work proposed a method to assist manual design of a single component of such flat-foldable model, consisting of vertically-connected side panels as well as horizontal top and bottom panels. Our method semi-automatically generates a more complicated model that approximates the input mesh with multiple convex components. The user specifies the folding direction of each convex component and the fidelity of shape approximation. Given the user inputs, our method optimizes shapes and positions of panels of each convex component in order to make the whole model flat-foldable. The user can check a folding animation of the output model. We demonstrate the effectiveness of our method by fabricating physical paper prototypes of flat-foldable models.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

1. Introduction

Folding objects flatly is an effective way to conserve storage space. For example, packing materials such as card-board boxes or gift boxes are devised to be flat-foldable and have their shapes restored easily. In the field of origami engineering, folding mechanisms have been extensively studied and used to develop various products that require contractility and portability, from pop-up cards to solar panels of satellites.

Although previous folding techniques are used in various fields, they can handle only relatively simple shapes such as symmetric ones. Additionally, complicatedly-shaped objects consisting of rigid materials are more difficult to fold than objects made of soft materials such as cloth or rubber. Also, the more complicated the shape is, the more difficult it is to fold and restore its shape.

To tackle this challenge, Kase et al. proposed a system to assist manual design of flat-foldable polygonal models [KKM15]. Figure 1 shows an example model designed by their system, which is defined by the shapes of the top panel and cross-section polylines specified by the user. The cross-section polylines define the side panels that are vertically connected each other and also connected to the top and bottom panels. We can fold the model flat by pushing down the top panel because horizontally-adjacent side panels are not connected and vertically-connected side panels are articulated at each horizontal edge that works as a hinge.

Unfortunately, their system has several problems. First, because their system is basically for manual design, the user must undergo many trial-and-errors to design a desired shape. Second, the designing process is not intuitive because the user specifies the shapes of the top panel and cross-section polylines in 2D space, making it hard to imagine the resultant shape of the whole model. Third, although they suggested that a complicated model can be created by designing and attaching multiple flat-foldable components each other, the user must account for which faces to be attached while adjusting the shapes and positions of faces and avoiding collisions during folding.

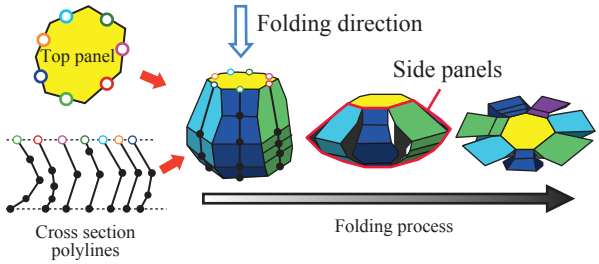

Figure 1: *Flat-foldable component by Kase et al. [KKM15].*

In this paper, we propose a semi-automatic method for creating a multi-component, flat-foldable model that roughly approximates an input 3D mesh with rigid panels. Our method first divides the input mesh into multiple components and, in order to simplify the shape optimization process, converts them into convex shapes. After the user specifies the folding direction of each convex component and fidelity parameters for shape approxima-

tion (e.g., the number of vertices in the top panel), our method calculates a flat-foldable shape for each component through optimization, while accounting for faces of inter-component attachment as well as collisions between horizontally-adjacent side panels in each component. The user can check a folding animation of the resultant model. To demonstrate the effectiveness of our method, we fabricate physical paper prototypes of flat-foldable models generated by our system.

2. Related Work

Techniques for folding objects flat have been used in various fields. A research topic familiar for the computer graphics community is to design pop-up cards or books, which are paper artworks that are flat while closed and show their 3D shape when opened [MS04, XTGH11, IEM*11, JLYL14]. The target objects of these study form their shape when being expanded 90 or 180 degrees, whereas each flat-foldable component stands up after being pulled perpendicularly to the top panel in our method. Additionally, we can make a complex object by combining multiple objects that are folded in different directions.

In the field of origami engineering, there are a number of techniques for designing flat-foldable shapes. Particularly, those for *rigid-foldable* shapes are the most relevant to our work. A rigid-foldable shape consists of rigid faces and mountain/valley crease lines, and the rigid faces remain rigid during folding, similarly to the rigid panels in our flat-foldable models. However, existing studies focus on how to fold simple primitives, e.g., cylindrical or cellular structures [TM12, YYT*13], unlike our work that handles general polygonal models using slits and multiple components.

For other foldable products, Li et al. presented a method for automatically generating foldable furniture [LHAZ15]. Koo et al. developed an interactive system to aid the design of work-alike prototypes of products to explore possible mechanical architecture [SSCO08]. They optimize model geometry on the basis of a functional relationship given by the user. While these studies focus on avoiding intersections of each component of an object to design foldable models, we focus on generating foldable objects that can be flattened out.

3. Configuration of Flat-Foldable Component

Before explaining our method, we briefly review the baseline method by Kase et al. [KKM15] to make this paper self-contained. After introducing the basic definitions, we explain their method and then describe our modification.

3.1. Basic Definitions of Flat-Foldable Component

As shown in Figure 1, a flat-foldable component is specified with a horizontal top panel as well as cross-section polylines. Each cross-section polyline is assigned to each top-panel edge; the cross-section polylines are as many as the top-panel edges. We can generate a vertical chain of side panels by extruding each cross-section polyline infinitely in parallel to the corresponding top-panel edge, followed by trimming with horizontally-adjacent side panels. The side panels are folded outward during folding. The top panel is a

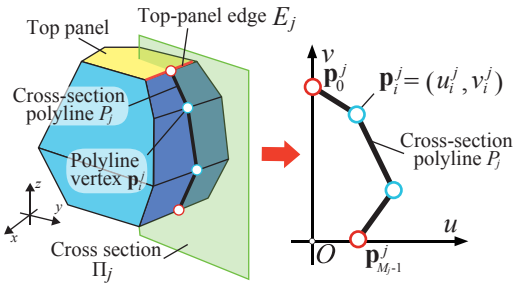

Figure 2: *Basic definitions of a flat-foldable component.*

planar polygon perpendicular to the folding direction. To avoid collisions of horizontally-adjacent side panels, the top panel must be convex [KKM15]. The bottom panel is parallel to the top panel, and automatically determined as a result of trimming with side panels.

Figure 2 summarizes the basic definitions. Suppose there are N edges on the top panel. For each top-panel edge E_j ($j = 0, 1, \ldots, N-1$), we consider cross section Π_j of the flat-foldable component, and then define cross-section polyline P_j on plane Π_j. Polyline P_j consists of vertices $\mathbf{p}_i^j$ ($i = 0, 1, \ldots, M_j - 1$), where M_j is the number of vertices in polyline P_j. Each vertex $\mathbf{p}_i^j$ corresponds to a horizontal edge connecting vertically-adjacent side panels. Polyline P_j is further examined in uv plane defined on cross section Π_j. We denote $\mathbf{p}_i^j = (u_i^j, v_i^j)$ in uv plane. $\mathbf{p}_0^j$ and $\mathbf{p}_{M_j-1}^j$ contact the top and bottom panels, respectively. $\mathbf{p}_0^j$ must move only along v axis, $\mathbf{p}_{M_j-1}^j$ is fixed on u axis during folding. Hereafter we might omit subscript j for simplicity.

3.2. Baseline Method by Kase et al.

Kase et al. [KKM15] examined the conditions that each polyline must satisfy to generate a flat-foldable component (see Figure 3):

$$u_i' \geq u_i, \tag{1}$$

$$\epsilon = (u_{M-1} - u_0) - \sum_i \delta_i l_i \approx 0, \tag{2}$$

where u_i and u_i' correspond to u coordinates of $\mathbf{p}_i$ before and after folding. l_i ($i = 0, 1, 2, , \cdots, M-2$) is the length of a segment S_i connecting $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$. ϵ represents the gap between u coordinates of the two endpoints in the flat-folded state. If gap ϵ of the polyline is sufficiently close to zero, the polyline is considered as flat-foldable. δ_i becomes 1 if S_i faces upward (i.e., the outward normal of a side panel corresponding to segment S_i is oriented upward), and -1 if downward. Each vertex has a folded or non-folded state, and the folded state has two folding configurations, i.e., mountain fold or valley fold. That is, the total number of possible folded configurations is 2^{M-1}. To obtain a good folded configuration, their system ranks each folded configuration using a score function, and selects the one with the best score. The system then optimizes the vertex positions in the selected configuration in order to make the polyline flat-foldable.

They also suggested to combine multiple flat-foldable components to create a more complicated model. Note that we must carefully select components' faces to be attached in order to avoid col-

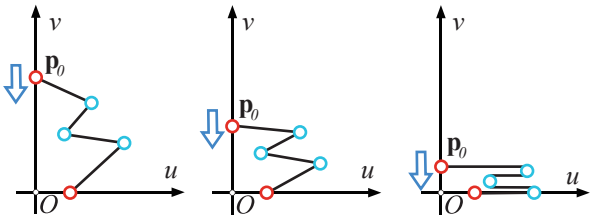

Figure 3: *Folding sequence of a cross-section polyline.*

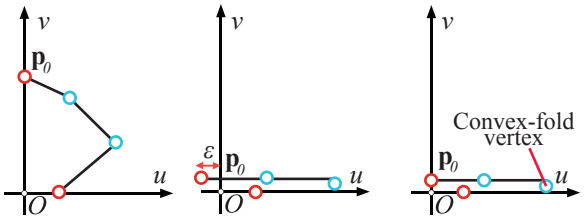

Figure 4: *Folding of a cross-section polyline with our convexity constraint. The gap ϵ (middle) in the folded state is minimized via optimization (right).*

lisions during folding. Figure 5 illustrates an example. As their system allows concave shapes, component B in Figure 5 has a concave component (with segments in green) where we cannot attach other components to avoid collisions.

3.3. Convexity Constraint in Our Method

Unlike the manual design in their system, we determine the whole shape of a flat-foldable component as automatically as possible while satisfying flat-foldability and shape fidelity. In order to make the problem tractable, we introduce a novel constraint that each polyline should have no valley fold and only one mountain fold in the flat-folded state, which ensures the resultant component has a convex shape. The only vertex having a mountain fold will move outermost in the flat-folded state, and we call it the *convex fold vertex* (see Figure 4, right). This constraint simplifies the problem:

1. All unconstrained vertices in each polyline always move outward in u direction and thus Eq. (1) is always satisfied. What we have to consider for flat-foldability is only Eq. (2).
2. When attaching multiple components, we do not have to consider concave components (Figure 5), unlike the manual design using the system by Kase et al.
3. The number of possible folding configurations is greatly reduced from 2^{M-1} to $M-2$ (note that we cannot assign a mountain fold to $\mathbf{p}_0$ or $\mathbf{p}_{M-1}$).

This convex constraint also makes the manual folding process easier; if we allow a concave component and push it down vertically, concave side panels will move inward and collide with adjacent side panels. To avoid such collisions, we have to manually pull out each concave side panel outward during the folding. This is much more troublesome in case of multiple concave components.

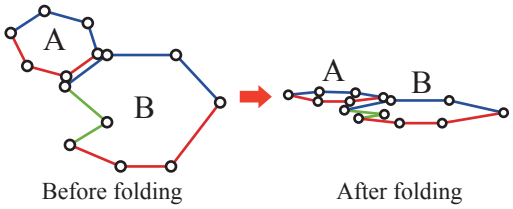

Figure 5: *2D illustration of component attachment. The green segments in component B indicate a concave component where other components cannot be attached to avoid collisions.*

4. Proposed Method

Figure 6 illustrates the overview of our method, which involves the following five steps. Namely, our method

1. Divides an input 3D mesh model into multiple components and converts them into convex shapes (Section 4.1),
2. Generates a top panel and optimizes its size and position for each component (Section 4.2),
3. Generates cross-section polylines and adjusts the number and positions of vertices of each polyline so that each polyline becomes foldable (Section 4.3),
4. Generates a 3D polygonal model from the top panel and cross-section polylines and then modifies the cross-section polylines again as needed (Section 4.4), and
5. Connects foldable components with one another.

Our method generates a flat-foldable model automatically, except for the user-specified folding directions and parameters for shape approximation fidelity.

4.1. Segmenting Mesh and Calculating Convex Components

To approximate an input 3D mesh with multiple flat-foldable components, we apply the feature-preserving mesh segmentation algorithm by Shapira et al. [SSCO08]. The top panel of each component must be convex (Section 3.1), and the cross-section polylines should not have valley folds causing inward move (Section 3.3), which makes the whole component shape convex. We therefore replace each segmented mesh with its convex hull calculated using [BDH96].

4.1.1. Determining folding direction

An important step to generate each flat-foldable component is to specify its folding direction, which determines fundamental factors of shape design. First, the folding direction specifies the top panel so that the top panel becomes perpendicular to the folding direction. The top panel then determines not only top-panel edges but also corresponding cross-section polylines, for which we have to minimize gaps in Eq. (2).

The shape of the top panel also affects inter-component attachment. Figure 7 illustrates the influence of the folding direction of a flat-foldable component, showing two resultant models with different folding directions specified for the teddy bear's face. The model in the upper row is created using the folding direction of the vertical direction from up to down. This folding direction specifies the topmost face as the top panel, and the ears are attached flatly. On

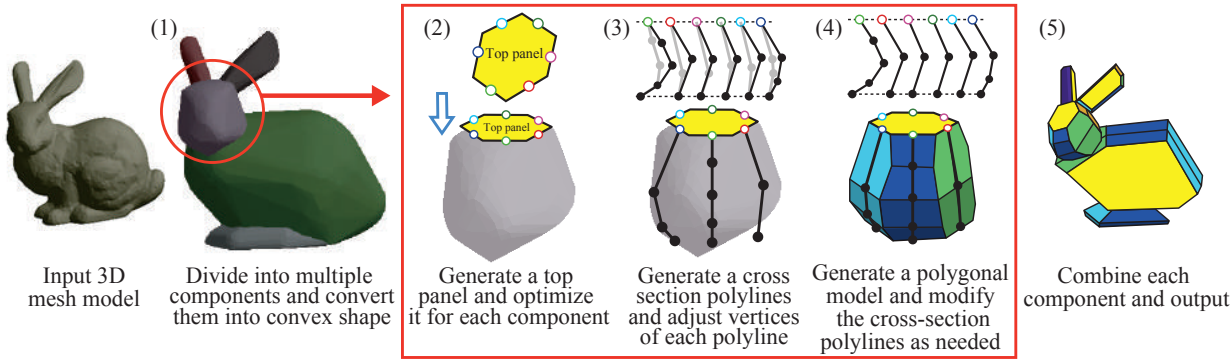

Figure 6: *Overview of our method. We first (1) segment an input 3D mesh and replace each mesh segment with its convex hull, which will be converted into a flat-foldable component. We then generate and optimize (2) a top panel and (3) cross-section polylines for each component. Next, we (4) generate a flat-foldable component from the top panel and cross-section polylines and then modify the polylines again as needed. Finally, we (5) combine flat-foldable components and output a foldable model.*

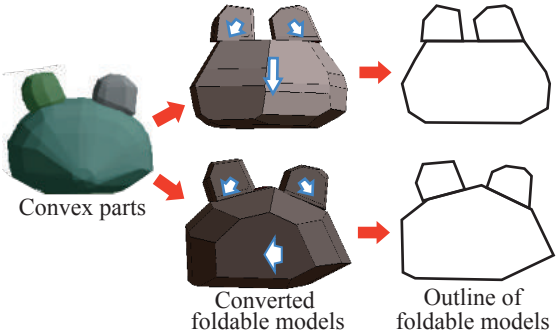

Figure 7: *Influence of the folding direction of a flat-foldable component. The different folding directions (white arrows outlined in blue) yield different shapes of flat-foldable models.*

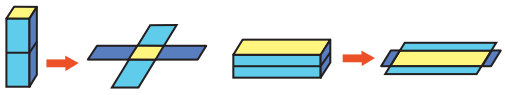

Figure 8: *Folding direction with a larger top panel (right) yields smaller spread after folding than that with a smaller top panel (right), resulting in less collisions during folding.*

the other hand, the other model in the lower row is created using the folding direction from back to front. In this case, the top panel is not visible in front view, but the bottom panel is visible as the flat polygon. Note that the ears are attached in different orientations.

As a criteria to determine the folding direction of each component, we consider the fact that a folding direction with a larger top panel yield smaller spread in the flat-folded state (Figure 8), which is beneficial to less collisions with other components during folding. We thus let the user specify a folding direction perpendicular to the first principle direction of the component shape.

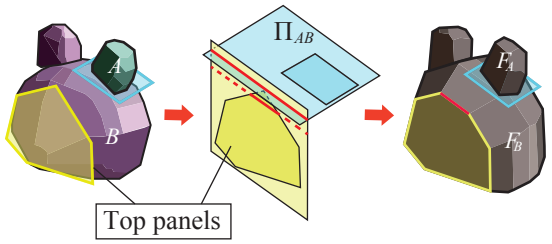

Figure 9: *Trimming B's top panel to ensure a side panel parallel to the connecting plane Π_{AB} for attaching component A.*

4.2. Generating Top Panel

Given the user-specified folding direction, we first generate the initial shape of the top panel, and then optimize its size and position. The initial shape of the top panel is generated as follows. Our method

1. Projects vertices of the convex component to a plane perpendicular to folding direction and convert them to points in 2D space,
2. Generates a 2D convex hull from the 2D points, and
3. Reduces vertices (and resp. edges) of the polygon up to the user-specified number, N^{min}.

For the edge reduction in step 3, we use the algorithm by Dyken et al. [DDS09].

4.2.1. Trimming top panel for attachment

When designing the top panel, we trim the top-panel polygon to generate side panels for attaching other flat-foldable components. Figure 9 illustrates an example. Let A and B be mesh segments and Π_{AB} be a plane fitted using least-squares to vertices located at their interface. To attach A to B, B should have a side panel parallel to plane Π_{AB}. Such side panel is defined by its cross-section polyline, and such polyline is specified by the corresponding top-panel edge. We therefore generate a top-panel edge parallel to plane Π_{AB} by trimming the top panel. If one of B's top-panel edges is almost parallel to plane Π_{AB}, we do not trim B's top panel but use such top-panel edge (and the corresponding side panel) of B. After ensuring

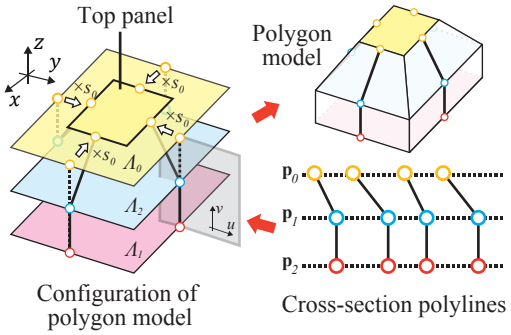

Figure 10: *Simplified model for optimizing the top panel. Polygons Λ_0,Λ_1, and Λ_2 are of the same shape as the initial top panel, and contain $\mathbf{p}_0^j$, $\mathbf{p}_1^j$, and $\mathbf{p}_2^j$ of cross-section polylines, respectively. Scales s_0, s_1, and s_2 as well as other parameters are optimized to obtain the polygon model shown in upper right.*

B's side panel for attaching A, we simply rotate and translate A to attach A onto B.

Note that the attached bottom panel of A might be too larger/smaller than B's side panel. In that case, the user re-designs the shapes by tuning the shape fidelity parameters (see Section 5.1).

4.2.2. Optimizing top panel

To optimize the size and position of the top panel, we also have to consider how well the corresponding cross-section polylines (i.e., centerlines of side panels) approximate the initial component shape. We simplify this problem by separating the optimizations for the top panel and cross-section polylines; here we optimize only the top panel with quite simplified cross-section polylines that roughly approximate the component shape, and later, with the fixed top panel, optimize cross-section polylines (Section 4.3).

Figure 10 illustrates the simplified model for optimizing the top panel. In this model, each cross-section polyline has only three vertices and two edges, and vertex $\mathbf{p}_i^j = (u_i^j, v_i^j)$ $(i = 0,1,2)$ in each cross-section polyline P_j has the same v coordinate. Let Λ_0, Λ_1, and Λ_2 be polygons that are parallel copies of the initial top panel and contain $\mathbf{p}_0^j$,$\mathbf{p}_1^j$, and $\mathbf{p}_2^j$, respectively.

To roughly approximate the initial component shape with this simplified model, we optimize the following eight parameters.

1. Scales s_0, s_1, and s_2 for each u-coordinates u_0^j, u_1^j, and u_2^j (3 parameters).
2. v coordinates v_0^j, v_1^j, and v_2^j (3 parameters).
3. x,y coordinates of the centroid of the simplified model (2 parameters).

Note that we omit the z coordinate of the centroid because the height is already considered with the v coordinates.

Let V_0 be the set of vertices in the initial component shape, and V be the set of vertices of the simplified model. We optimize the eight parameters by minimize the following energy function:

$$E = \sum_{a \in V_A} \min_{b \in V_B} dist(a_s, b) + \sum_{b \in V_B} \min_{a \in V_A} dist(b_t, a) \qquad (3)$$

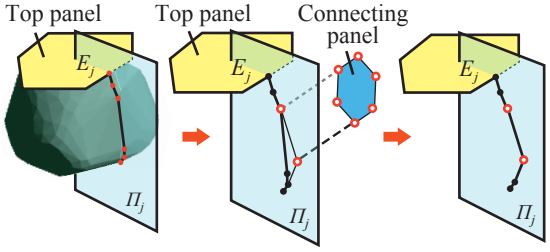

Figure 11: *Generating an initial polyline by connecting intersections (red) of the convex hull with cross section Π_j corresponding to the top-panel edge E_j (left). To attach another component, we insert an edge into the polyline (right), which is parallel to the component's panel to be attached.*

$$s.t. \quad v_0^j > v_1^j > v_2^j, \qquad (4)$$

$$0 < s_i \leq 1, \qquad (5)$$

$$H_{min} \leq v_2^j,\ v_0^j \leq H_{max}. \qquad (6)$$

Eq. (3) is the sum of the shortest distances between the initial component shape and the simplified model, approximating the exclusive OR of their volumes. Constraint (4) is to arrange vertices consistently in v direction. The condition $0 < s_i$ in constraint (5) prevents inversion in the simplified model. The condition $s_i \leq 1$ is to shrink polygon Λ_i from its initial size; polygon Λ_i initially has the same shape as the top panel and is expected to shrink. Constraint (6) is to keep the height of the simplified model within the specified range, where H_{min} and H_{max} denote the lower and upper bounds of the heights of the top and bottom panels along the folding direction. Note that foldability of cross-section polylines is omitted in this optimization because here we focus on appropriate size and position of the top panel. We consider the foldability of cross-section polylines in the next subsection.

4.3. Generating Cross-section Polylines

Given the top panel optimized in the previous subsection, we obtain initial cross-section polylines as follows. For each cross section Π_i passing through the midpoint of top-panel edge E_j, we calculate intersection points of edges on the convex component, and use them as the initial set of vertices of polyline P_j (Figure 11, left). Because we seek for convex components, vertices in the initial polylines are removed if they do not contribute to convex hulls.

Additionally, we also have to consider inter-component attachments, similar to the discussion in Section 4.2.1. To attach a component, we need a side panel that is parallel to the target component's panel to be attached. We thus insert a line segment that defines such side panel to the cross-section polyline P_j. Specifically, we project the vertices of the attaching panel onto cross section Π_j, and insert a segment covering the projected vertices (Figure 11, right). Note that, if there are no projected vertices within the range of the top and bottom panels, we do not insert a line segment, e.g., in the case that the attaching panel is parallel to the top panel.

4.3.1. Optimizing polylines independently

This section describes how to optimize each cross-section polylines to be foldable independently, without considering other poly-

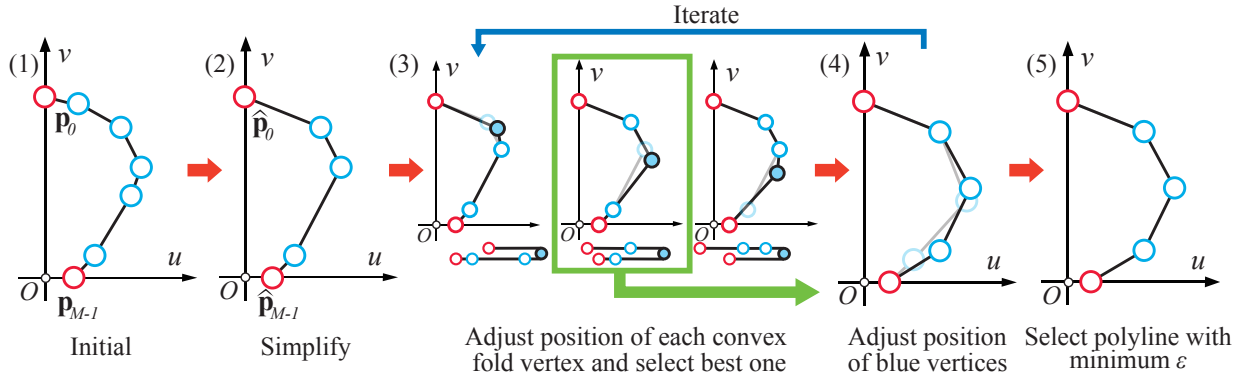

Figure 12: *Optimization of a cross-section polyline.* $\mathbf{p}_i$ $(i = 0,1,2,\cdots,M-1)$ *and* $\hat{\mathbf{p}}_{i'}$ $(i' = 0,1,2,\cdots,k-1)$ *are vertices before and after simplification. (1) We remove vertices of the polyline until the number of vertices becomes* k $(3 \le k < k^{max})$*. We then (2) adjust the position of the convex-fold vertex and select the polyline with the smallest gap* ϵ*, and (3) adjust the positions of all vertices. If* $k < k^{max}$*, we return to Step (3).*

lines. The basic procedure is to generate candidate polylines by simplification and optimization, and to select a simplified polyline that has the smallest gap ϵ among the candidates. The details are illustrated in Figure 12 and explained as follows.

1. We generate candidate polylines by simplifying the initial polyline P_j with different numbers of vertices. Let k be the number of vertices in a simplified polyline $\hat{P}_j^k$. Note that $\mathbf{p}_0$ and $\mathbf{p}_{M-1}$ are fixed, and vertices belonging to the line segment parallel to the connecting plane are only allowed to move in parallel to the plane. We iterate the following steps from $k = 3$ to $k = k^{max}$, where k^{max} is the maximum number of vertices specified by the user.
2. For each simplified polyline $\hat{P}_j^k$, we pick each vertex $\mathbf{p}_h$ $(h = 1,2,\cdots,k-2)$ as a candidate of a convex-fold vertex, optimize $\hat{P}_j^k$, and then examine the gap ϵ. Here we optimize the uv coordinates of only $\mathbf{p}_h$ using Eq. (7) with a constraint $v_{h+1} < v_h < v_{h-1}$ to avoid vertical flip. We then determine $\mathbf{p}_h$ with the smallest gap ϵ as the convex-fold vertex of simplified polyline $\hat{P}_j^k$.
3. For each simplified polyline $\hat{P}_j^k$ with different k, we optimize the whole shape of $\hat{P}_j^k$ using Eq. (7) with the constraint (11) and examine the gap ϵ. Again, $\mathbf{p}_0$, $\mathbf{p}_{M-1}$ and vertices belonging to the line segment parallel to the connecting plane are constrained as Step 1. Finally, we select $\hat{P}_j^k$ with the smallest gap ϵ as the output of this procedure.

To simplify a polyline in Step 1, we used the Douglas-Peucker algorithm with a slight modification; we reduce vertices until the number of points becomes less than the specified number. The purpose of adjusting only one convex fold vertex in Step 2 is to choose more appropriate folding configuration before optimization of all vertices, and to suppress the influence of the change of the shape as much as possible in only the convex fold vertex.

We define an energy function by considering foldability and fidelity of the simplified cross-section polyline to the initial polyline, and adjust positions (i.e., u and v coordinates) of vertices by minimizing the energy function as follows.

$$E = w_s E_s + w_g E_g + w_c E_c, \tag{7}$$

where E_s, E_g and E_c are energy terms described hereafter, and w_s, w_g, and w_c are the weights. We empirically choose $w_s = 0.7$, $w_g = 1.0$, and $w_c = 0.9$.

Shape fidelity term. To prevent an adjusted polyline from differing from original shape during optimization, we add the following term:

$$E_s = d_H(\{\mathbf{p}_i\},\{\hat{\mathbf{p}}_{i'}\}), \tag{8}$$

where $d_H(\mathcal{A},\mathcal{B})$ denotes Hausdorff distance between points set $\mathcal{A}$ and $\mathcal{B}$, and $\mathbf{p}_i$ $(i = 0,1,2,\cdots,M-1)$ and $\hat{\mathbf{p}}_{i'}$ $(i' = 0,1,2,\cdots,k-1)$ are the vertices before and after simplification.

Foldability term. We regard a cross-section polyline as flat-foldable if its gap ϵ is less than a tolerance τ. Similarly to the baseline method [KKM15], we use $\tau = 0.001 \times \bar{l}$, where $\bar{l}$ is the average segment length of the cross-section polyline. To minimize gap ϵ, we add the following term:

$$E_g = \left| u_{M-1} - u_0 - \sum_{i=0}^{h-1} l_i + \sum_{i=h}^{M-1} l_i \right|, \tag{9}$$

where h is an index of a convex fold vertex of the cross-section polyline.

Convexity term. The cross-section polyline should be as convex as possible so that it is flattened outward. To measure the convexity of the line, we define a new term that is the sum of distance between an edge $\mathbf{p}_{i-1}\mathbf{p}_{i+1}$. If it equals zero, the shape of the polyline is convex to outward. The new term is:

$$E_c = \sum_{i=1}^{M-2} d_i, \qquad d_i = \begin{cases} 0 & if\ \ \theta_i \le \pi \\ dist(\hat{\mathbf{p}}_{i-1}\hat{\mathbf{p}}_{i+1}, \hat{\mathbf{p}}_i) & otherwise \end{cases} \tag{10}$$

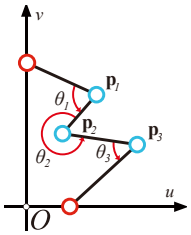

Figure 13: *Angles θ_i are calculated counterclockwise.*

where $dist(E, \mathbf{p})$ is the Euclidean distance between edge E and point $\mathbf{p}$. θ_i is a counterclockwise angle between vectors $\hat{\mathbf{p}}_{i+1} - \hat{\mathbf{p}}_i$ and $\hat{\mathbf{p}}_{i-1} - \hat{\mathbf{p}}_i$ (see Figure 13).

To avoid inconsistent ordering in the vertical direction, we minimize Eq. (7) with the following constraint:

$$v_{i+1} < v_i < v_{i-1}. \tag{11}$$

Note that there are cases where the gap ϵ does not become less than τ. Thus, we iterate optimization while decreasing w_1 and w_3 until gap ϵ becomes less than τ because foldability is more important than shape fidelity and convexity.

To solve this optimization problem, we use gradient descent with numerical differentiation, starting from the initial shape that satisfies all the constraints. If we violate constraints during an iteration, we halve the step size and re-evaluate the iteration. Particularly, if we violate constraint (11) at vertex i, we replace v_i with a value slightly smaller than v_{i-1}. To keep the direction of the line segment parallel to the connecting plane, the vertices consisting of the line segment are constrained to move only along that direction. This is accomplished by parameterizing the endpoint with a scalar value and a unit directional vector, and by optimizing the scalar value instead of u, v coordinates of the endpoint.

In the results shown in Section 5, our optimization always converged probably thanks to good initial shapes, and yielded flat-foldable models mainly because we choose a flat-foldable configuration among several candidates as in Step 2.

4.4. Optimizing Polylines While Avoiding Collisions with Neighbors

In Section 4.3 we optimized each cross-section polyline separately. However, the side panels may collide with each other depending on shape of the adjacent cross-section polylines, and thus we cannot generate foldable polygonal components sometimes. Therefore, to generate a foldable polygonal model properly, this section describes a method to modify the vertex positions of the entire polylines by considering relationship of vertices of the adjacent cross-section polylines.

Here, we explain the collision of the side panels by using Figure 14. First, we define some symbols. Let c_i^{j+1} in the P_{j+1} is a point that have same v-coordinate as $\mathbf{p}_i^j$.

To explain the collision, we consider two lines; the one line L_{j+1}^i includes c_{j+1}^i and is parallel to top-panel edge E_{j+1}, and the other line L_j^i includes $\mathbf{p}_i$ and is parallel to top-panel edge E_j. Letting $\mathbf{m}_j$

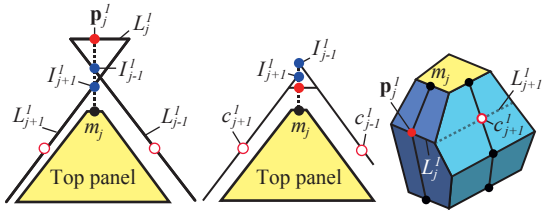

Figure 14: *Top views of an invalid mesh (left) as well as top (middle) and oblique (right) views of a valid mesh.*

be the midpoint of top-panel edge E_j, we can define I_{j+1}^i as the intersection of a line $\mathbf{p}_j^i\mathbf{m}_j$ and L_{j+1}^i. This is shown in middle of Figure 14.

The conditions under which collision occurs are described as:

$$\|\mathbf{p}_j^i - \mathbf{m}_j\| - \|I_{j+1}^i - \mathbf{m}_j\| > 0, \tag{12}$$

$$\|\mathbf{p}_j^i - \mathbf{m}_j\| - \|I_{j-1}^i - \mathbf{m}_j\| > 0, \tag{13}$$

where, because I_{j+1}^i and I_{j-1}^i are on the same line, the condition under which collision does not occur is defined as:

$$\|\mathbf{p}_j^i - \mathbf{m}_j\| - \min(\|I_{j+1}^i - \mathbf{m}_j\|, \|I_{j-1}^i - \mathbf{m}_j\|) \le 0. \tag{14}$$

The absolute value of the left side of Eq. (14) is equal to $\max(\|I_{j+1}^i - \mathbf{p}_j^i\|, \|I_{j-1}^i - \mathbf{p}_j^i\|)$. Therefore, to generate foldable polygonal components properly, we modify the polylines by minimizing the energy function with an additional objective function:

$$E_f = \sum_{j=0}^{N-1} \sum_{i=1}^{M-2} \max(\|I_{j+1}^i - \mathbf{p}_j^i\|, \|I_{j-1}^i - \mathbf{p}_j^i\|), \tag{15}$$

and we update Eq. (7) to the following energy function:

$$E = w_s E_s + w_g E_g + w_c E_c + w_f E_f, \tag{16}$$

where we used the weights $w_s = 0.01$, $w_g = 1.0$, $w_c = 0.01$, and $w_f = 1.0$. We optimize Eq. (16) for each polyline until convergence.

5. Results

We implemented our algorithm using C++, and ran our program on a PC with an Intel Core i7 and 8GB RAM. We used the CGAL C++ library for segmenting an input 3D mesh model and converting segments into multiple convex components.

Figure 18 shows the foldable polygonal models generated by using our system, and Figure 19 shows the animation of folding these models. Figure 20 shows the fabricated physical paper prototypes from development planes. For physical assembly, we used tapes and bond for attachment. These results demonstrate that our system can convert input 3D models into foldable models, and the converted models can be folded in the real world. Table 1 shows the computation time. The segmentation includes a process of dividing and converting to multiple convex components. In addition, the conversion of foldable model includes generation and optimization of the top panel, generation and optimization of cross-section polylines. According to Table 1, the segmentation takes longer time when the number of vertices is larger. This is because it takes time to convert to convex shape rather than segmentation. On the other

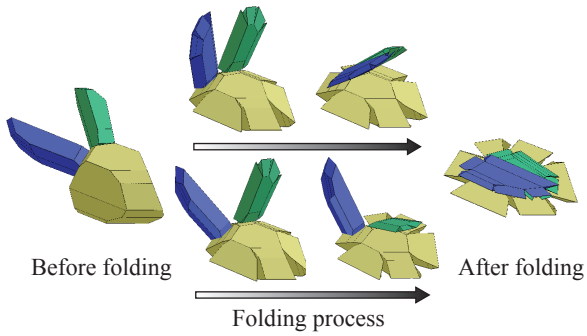

Figure 15: *Collision might occur if we fold multiple components simultaneously (upper row) but we can usually avoid collision by folding each component one by one (lower row).*

hand, although the number of models and the number of vertices affect calculation time, there is not much difference in calculation time for each foldable component.

Some flat-foldable models require attention to avoid collisions during folding. For example, when we fold the bunny's head model in Figure 15, the ears might collide with each other if we fold them simultaneously (upper row). However, collisions do not occur if we fold each ear one by one (lower row). In case that self-collision is unavoidable (we did not experience this situation in our results; collisions are usually avoidable as in Figure 15) or the resultant shape is unfavorable, the user can re-design the model with additional interactions, as described in the following subsection.

5.1. Influence of User Interactions

Our method is automatic except for the following user interactions; the user specifies the folding direction of each flat-foldable component (Section 4.1.1) and tunes the shape fidelity parameters, i.e., N^{min} (Section 4.2) and k^{max} (Section 4.3.1). As GUIs, the user specifies the folding direction by rotating it around the principle shape axis using the mouse wheel, and N^{min} and k^{max} using spinners, respectively. The folding direction can be specified for each component independently, as demonstrated in Figures 7 and 19 as well as the accompanying video. If the user-specified folding direction is almost parallel to the current connecting plane (Figure 9), we quit using the connecting plane and instead use the top or bottom panels as a new connecting plane.

Shape fidelity can be controlled by changing N^{min} and k^{max}. Table 2 summarizes the Hausdorff distance as an error metric between the resultant shape and the initial convex shape of each component of Bunny. The Hausdorff distances are normalized by the diagonal length of Bunny's bounding box. With larger N^{min} and k^{max}, the Hausdorff distance decreases but the difficulty in flat folding increases because of the more top-panel edges and more side-panel hinges.

5.2. Comparison with Kase et al. [KKM15]

Figure 16 shows a comparison of our method and the method by Kase et al. [KKM15] in terms of quality and time required for design. Our result (middle) was created semi-automatically from the

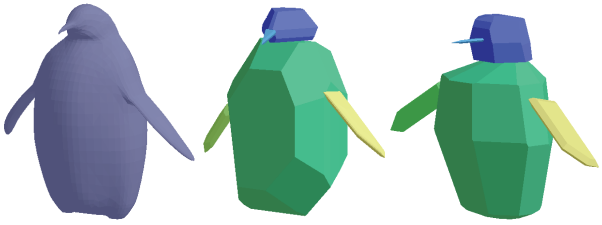

Figure 16: *Comparison of our result (middle) created semi-automatically from the input mesh (left) and a manually-created model (right) using the method by Kase et al. [KKM15].*

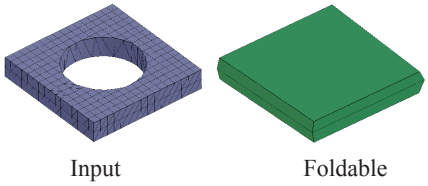

Figure 17: *The hole in the input mesh (left) is lost in the flat-foldable convex component (right).*

input mesh (left) while their result (right) was manually designed with a moderate effort to make the result resemble the original. The time required for designing the models with several trial-and-errors are about 20 minutes and 50 minutes, respectively. In general, while the method by Kase et al. [KKM15] provides much greater degrees-of-freedom in the designing process, our method yields plausible results in a shorter time.

Our design restriction that each flat-foldable component should have a convex shape (Section 3.3) affects the resultant shapes. For example, geometric details such as Bunny's eyes or ears are lost in Figure 18. Even worse, if the target polygonal model has a hole, the hole will be lost in its convex hull (Figure 17). This issue can be alleviated by increasing the number of segmented components, with increased difficulty in the shape design and physical realization.

6. Conclusions and Future Work

We have proposed a system that converts input 3D shape into foldable shape of polygonal models semi-automatically. The user specifies folding direction of each divided convex component from directions perpendicular to a first principle component of the vertices of the convex component, and the minimum number of edges of a top panel and the maximum number of cross-section polylines. We demonstrated that our system can generate foldable polygonal models.

As the most interesting future direction, we would like to loosen the convex shape constraint for each component (Section 3.3) in order to increase the shape approximation fidelity. Although allowing concave components will increase the designing and folding labor, the labor might be alleviated if we can actuate a foldable model mechanically [KMM17].

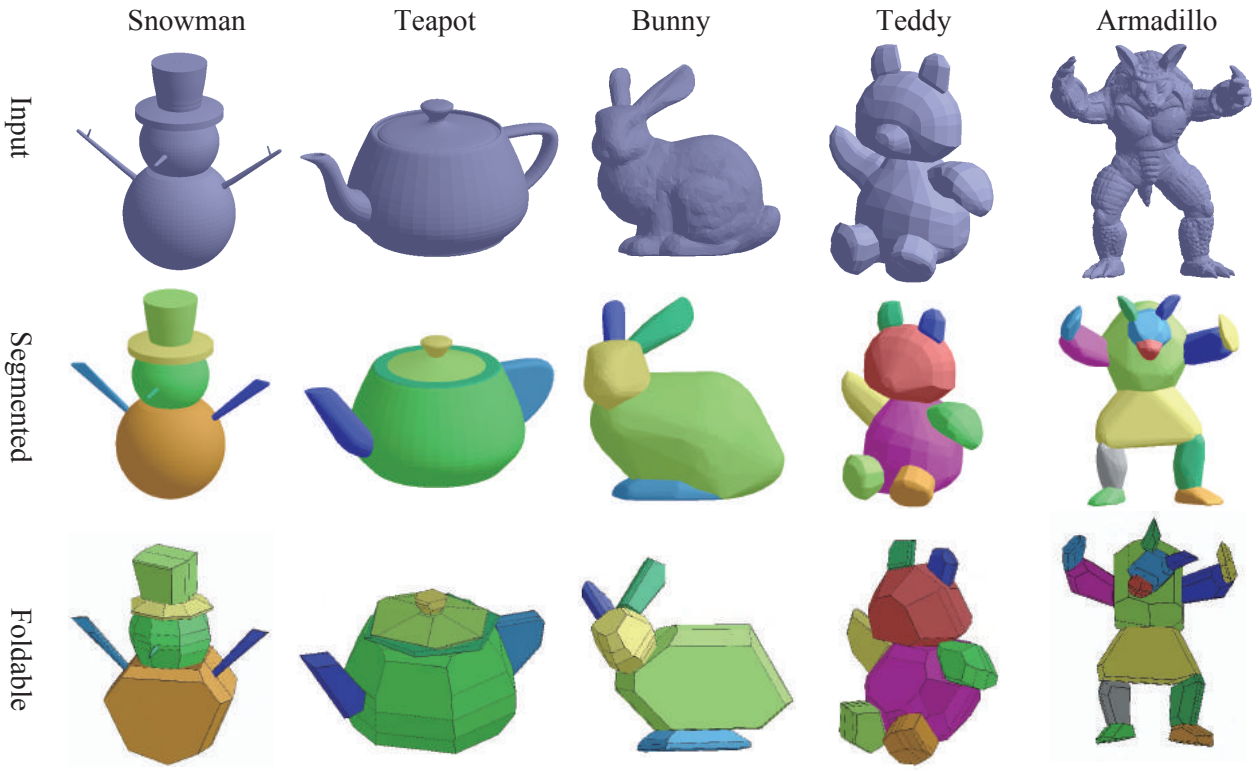

Figure 18: *Examples of input models and resultant flat-foldable models. Left column: Input models. Middle column: Convex hulls of segmented meshes. Right column: Flat-foldable components.*

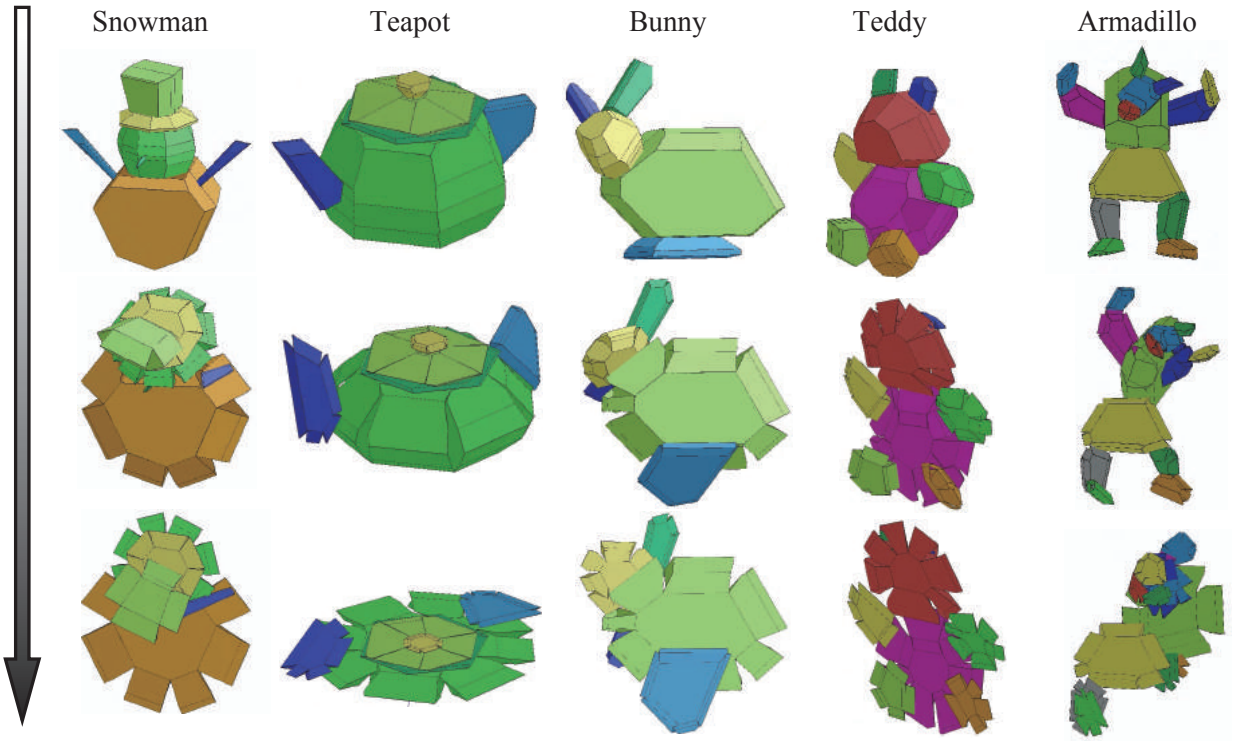

Figure 19: *Example animation sequences of flat-foldable models.*

Table 1: *Statistics of resultant flat-foldable models.*

	# of input faces (# of components)	Segmentation (sec)	Conversion of foldable model (sec)	Total time (sec)
Snowman	13,450 (6)	771	234	1,005
Teapot	6,543 (6)	148	193	341
Bunny	4,968 (5)	13	194	207
Teddy	3,192 (8)	27	174	201
Armadillo	17,296 (15)	63	449	512

Table 2: *Approximation errors of Bunny's components w.r.t. N^{min} and k^{max}.*

		Hausdorff distance (# of vertices of top panel)					
N^{min}	k^{max}	Left ear	Right ear	Head	Body	Foot	average
5	6	0.048 (5)	0.038 (5)	0.056 (5)	0.092 (5)	0.098 (5)	0.066
5	10	0.046 (5)	0.034 (5)	0.063 (5)	0.087 (5)	0.097 (5)	0.065
8	6	0.049 (8)	0.052 (8)	0.039 (8)	0.093 (8)	0.081 (8)	0.063
5	3	0.044 (5)	0.047 (5)	0.062 (6)	0.097 (5)	0.101 (5)	0.070
4	6	0.051 (4)	0.032 (4)	0.055 (5)	0.100 (5)	0.092 (4)	0.066

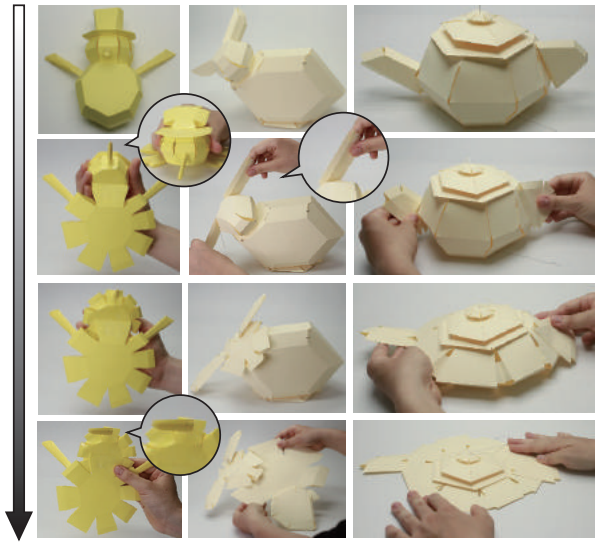

Figure 20: *Examples of physical paper prototypes of flat-foldable models.*

References

[BDH96] Barber C. B., Dobkin D. P., Huhdanpaa H.: The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software 22*, 4 (1996). 3

[DDS09] Dyken C., Dæhlen M., Sevaldrud T.: Simultaneous curve simplification. *Journal of Geographical Systems 11*, 3 (2009), 273–289. 4

[IEM*11] Iizuka S., Endo Y., Mitani J., Kanamori Y., Fukui Y.: An interactive design system for pop-up cards with a physical simulation. *The Visual Computer (Proc. of Computer Graphics International 2011) 27*, 6-8 (2011), 605–612. 2

[JLYL14] Jr C. R. R., Le S. N., Yu J., Low K.-L.: Multi-style paper pop-up designs from 3D models. *ACM Trans. on Graphics 33*, 2 (2014), 487–496. 2

[KKM15] Kase Y., Kanamori Y., Mitani J.: A method for designing flat-foldable 3D polygonal models. In *Mechanisms and Robotics Conference* (2015), pp. DETC2015–46566. 1, 2, 6, 8

[KMM17] Kilian M., Monszpart A., Mitra N.: String actuated curved folded surfaces. *ACM Transactions on Graphics 36*, 3 (2017). 8

[LHAZ15] Li H., Hu R., Alhashim I., Zhang H.: Foldabilizing furniture. *ACM Transactions on Graphics, (Special issue of SIGGRAPH Asia 2014) 34*, 4 (2015), 90:1–90:12. 2

[MS04] Mitani J., Suzuki H.: Computer aided design for origamic architecture models with polygonal representation. In *Proc. of CGI 2004* (2004), pp. 1372–1379. 2

[SSCO08] Shapira L., Shamir A., Cohen-Or D.: Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer: International Journal of Computer Graphics 24*, 4 (2008), 249–259. 2, 3

[TM12] Tachi T., Miura K.: Rigid-foldable cylinders and cells. *The International Association for Shell and Spatial Structures 53*, 4 (2012). 2

[XTGH11] X.Li, T.Ju, Gu Y., Hu S.: A geometric study of v-style pop-ups: Theories and algorithms. *ACM Transactions on Graphics 30*, 4 (2011), 98:1–10. 2

[YYT*13] Yasuda H., Yein T., Tachi T., Miura K., Taya M.: Compression behavior of tachi-miura polyhedra bellows. *Proc. of the Royal Society A 469*, 2159 (2013). 2