Pacific Graphics 2019
C. Theobalt, J. Lee, and G. Wetzstein
(Guest Editors)

*Volume 38 (2019), Number 7*

# Single-View Modeling of Layered Origami with Plausible Outer Shape

Y. Kato, S. Tanaka, Y. Kanamori and J. Mitani

University of Tsukuba, Japan

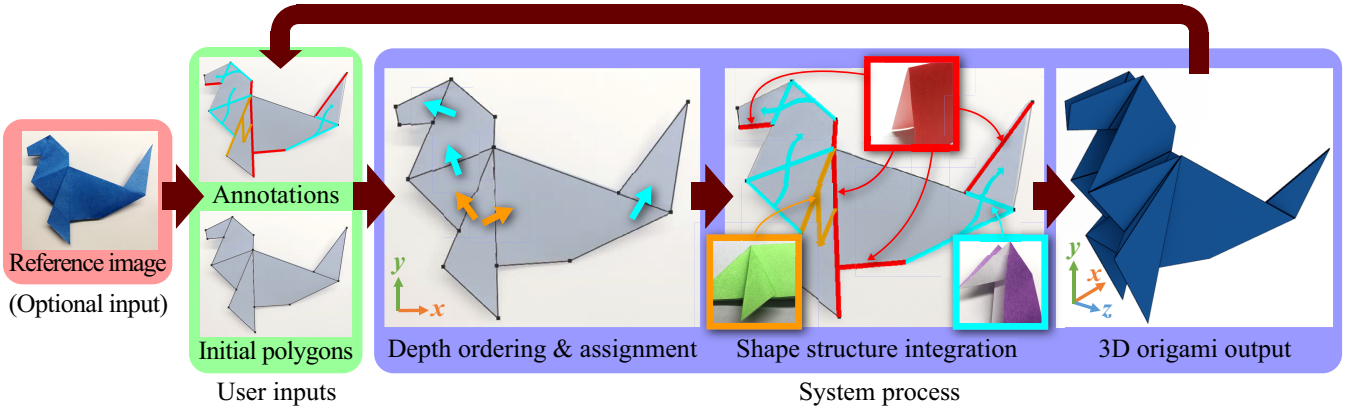



**Figure 1:** *Overview of our method. Given a reference image of a flat origami piece, the user draws polygons and assigns annotations indicating the types of folding. Our system then determines the depth order of polygons, assigns depth values to polygons, and integrates shape structures corresponding to the user-specified folding operations to output a 3D origami model with a plausible outer shape. The user can repeat this process with instant feedback until satisfied.*

**Abstract**

*Modeling 3D origami pieces using conventional software is laborious due to the geometric constraints imposed by the complicated layered structure. Targeting origami models used in visual content such as CG illustrations and movies, we propose an interactive system that dramatically simplifies the modeling of 3D origami pieces with plausible outer shapes, while omitting accurate inner structures. By focusing on flat origami models with a front-and-back symmetry commonly found in traditional artworks, our system realizes easy and quick modeling via single-view interface; given a reference image of the target origami piece, the user draws polygons of planar faces onto the image, and assigns annotations indicating the types of folding operations. Our system automatically rectifies the manually-specified polygons, infers the folded structures that should yield the user-specified polygons with reference to the depth order of layered polygons, and generates a plausible 3D model while accounting for gaps between layers. Our system is versatile enough for modeling pseudo-origami models that are not realizable by folding a single sheet of paper. Our user study demonstrates that even novice users without the specialized knowledge and experience on origami and 3D modeling can create plausible origami models quickly.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

## 1. Introduction

Making an origami piece by folding a sheet of paper is familiar worldwide as a recreation that even children can enjoy. However, modeling 3D origami pieces using conventional 3D software is laborious and time-consuming; theoretically, there are inherent geometric constraints w.r.t. origami modeling, i.e., an origami shape be homeomorphic to a disk because it is made of a single sheet of paper, and the sum of face angles around any vertex equal $2\pi$ because

an origami shape consists of developable surfaces. These conditions might be relaxed for the use in visual contents such as CG illustrations and movies because, in this case, accurate geometry is not necessary. However, a plausible outer shape matters instead; let us consider "origami animals" as an example. With a quick Google image/video search, we can see that they are popular motifs of illustrations/animations and often represented as mirror-symmetric flat shapes. Such *flat origami* can be completely flattened by definition if we ignore thickness, but, in the real world, even a flat origami piece has a certain thickness due to the small gaps between layers of sheets. Therefore, for modeling a realistic origami shape, we should handle such interlayer gaps while avoiding self-intersections between layers, which is quite tedious to do if we use conventional software.

Targeting flat origami with a front-and-back symmetry typified by popular origami animals, we propose an interactive system that dramatically simplifies the modeling of flat origami by omitting accurate inner shapes. The user can model plausible outer shapes easily and quickly via a single-view interface; given a reference image of the target origami piece, the user draws polygons of the planar faces onto the image and annotates the polygons to specify the types of folding operations using simple mouse gestures (see Figure 1). The user inputs polygons and annotations in a 2D-view window, and confirms the resultant 3D origami model in a 3D-view window with real-time feedback against user inputs.

Our system assists the user and calculates outer shapes automatically (Figure 1) from user inputs as follows. First, every time the user draws polygons, our system tries to adjust user-specified edges and vertices to existing ones for rectification (Section 5.1). Second, every time the user annotates polygon edges, our system automatically infers an appropriate folding operation that matches the annotation, which is the crucial feature of our system for efficient modeling. The folding operations supported in our system are commonly-used seven types of operations and a duplicating operation called *mirroring* (Section 3). The selected folding operation might induce geometric constraints to the configuration of input polygons, and thus our system checks whether such constraints are satisfied, and otherwise rectifies mis-aligned polygons automatically (Section 5.3). The selected folding operation also determines the local depth order among adjacent polygons. Our system maintains the depth order as a directed graph and updates it according to the annotations. By referring to the depth order, our system adds interlayer gaps to our origami model to form a realistic shape (Section 5.2). Finally, our system integrates the current origami model with the shape structure derived from the selected folding operation while resolving the depth order (Section 5.4).

In summary, the main contributions of this paper are as follows.

- The novel paradigm in the system design of origami modeling, i.e., forming plausible outer shapes quickly by omitting complicated inner structures,
- The single-view interface with simple annotations for modeling flat origami with a front-and-back symmetry, and
- The gap-aware modeling for realistic origami shapes based on the depth order of polygons.

Besides, because our system provides higher degrees-of-freedom in the shape design than conventional systems, the user can create even pseudo-origami models that are not realizable in the real world by folding a sheet of paper. By comparing with conventional software, we show that our system yields simpler yet plausible origami models with fewer polygons in much shorter time. Our simplified models are also advantageous in creating animations because self-intersections occur less frequently and less shape manipulations are required to specify keyframes. Our user study demonstrates that even novice users without the specialized knowledge and experience on origami and 3D modeling can create plausible origami models quickly.

## 2. Related Work

### 2.1. 3D modeling via 2D-based user interface

Modeling systems with a 2D user interface simplifies the 3D modeling of specific types of shapes. Sketch-based 3D modeling systems, e.g., [IMT99, NISA07], typically let the user draw 2D contours of the target shapes, and then the systems inflate the contoured regions to create organic shapes such as characters and animals. Whereas aforementioned 3D modeling systems enforce the user to change viewpoints frequently during modeling, single-view modeling (i.e., with a fixed viewpoint) is possible by focusing on specific targets, e.g., quadruped animals [EBC*15]. Bessmeltsev et al. [BCV*15] demonstrated that plausible 3D shapes containing occluded parts can be created from a single sketch of contours with the aid of a 3D skeleton and Gestalt principles. Gingold et al. [GIZ09] proposed a generic single-view modeling system where the user places generalized cylinders as shape primitives and several types of 2D annotations onto a reference image.

While these modeling systems target organic shapes, we focus on flat origami pieces consisting of layered structures of flat polygonal faces. Our system also allows single-view modeling with 2D annotations, but our annotations are specialized for specifying the folding operations.

Igarashi and Mitani [IM10] proposed an interesting method for manipulating the depth order of layered nearly-flat objects in a single view. The user can change the depth order of layers just by clicking or dragging a layer. Entem et al. [EPB*19] proposed a method that interprets a 2D sketch as elements layered in depth by making use of a graph giving local depth ordering information. Although their methods are similar to ours, they do not account for the folded shapes of origami.

### 2.2. Physically-based modeling of paper

An intuitive direction of modeling origami would be to simulate the physical behaviors of paper. Paper is non-stretchable, and the shapes made of paper consist of developable surface patches. There exist many methods for modeling or simulating developable surfaces [MYYT96, RSW*07, PDRK14, TBWP16]. Recent methods [NPO13] based on the finite element method can even handle not only folding but also crumpling of developable surfaces. A similar interactive method for crumpling paper [SRH*15] is mostly based on geometric modeling rather than finite element simulation. The folding mechanisms of rigid panels are discussed in the subarea called *rigid folding* in origami engineering and have

been studied extensively in parametric regular patterns, e.g., Miura-ori [GP94, YKD11, Sta12]. Miyamoto et al. [MEKM17] proposed a semi-automatic method for making a polygonal 3D shape flat-foldable by approximating it as multiple convex components with vertically-connected rigid panels. Alternatively, there is a method for reconstructing the folded shapes of paper with curved creases from point clouds [KFC*08]. Contrary to the aforementioned methods, our target shapes are neither rigid-foldable in general nor curved.

### 2.3. Origami shape modeling

Existing simulators of origami assist the user in simulating the sequence of real-world folding operations from an initial shape (usually a square) and have a long history of research [MYYT96, FKMF07]. Such simulators help the user reproduce folded shapes faithfully with a simple user interface. However, mouse-based manipulation of folds occurring simultaneously is intrinsically difficult compared to hand-based manipulation in the real world, and thus the user often struggles with modeling even flat origami models that we target in this paper.

There have been proposed methods for modeling flat origami. Lang's *TreeMaker* [Lan11] is one of the famous software for designing flat origami. Given a 2D skeleton of a target shape, it generates a crease pattern automatically but does not account for the depth order of faces. The software for editing crease patterns of flat origami, *ORIPA* [Mit05], has a feature for generating a shape after folding while accounting for the depth order. Ida et al. [IMKG09] developed a system for generating folded shapes based on formal expressions of folding processes. *Tess* [Bat07] is the software for designing *origami tessellations*, i.e., geometric tiling patterns that can be folded flat.

A problem with these methods is that their output shapes look unrealistic because they ignore thickness and thus become completely flat. In contrast, flat origami models in the real world have certain thickness, as mentioned in Section 1. A workaround is to manually add offsets to vertex positions with reference to the depth order of layered paper sheets while avoiding self-intersections, which is quite tedious. Additionally, the shapes generated by existing systems contain complicated inner structures, which are invisible and thus not necessary for the use in visual contents. Unnecessary inner structures increase the time required for modeling. Our system thus omits accurate inner structures and generates plausible outer shapes quickly.

Tachi [TAC10] proposed a method that outputs a single-sheet folding pattern from an arbitrary polyhedral surface. The folding patterns consist of not only the original facets but also tucks assigned to fill the gaps. His goal is to generate folding patterns that are physically foldable. In contrast, our system also supports origami models that are not physically foldable.

## 3. Preliminary: Operations Supported in Our System

In this section, we describe the seven types of folding operations and the duplicating operation supported in our system.

Our system supports flat origami with a front-and-back symmetry; the back shape is the mirror image of the front shape. This condition is commonly observed in traditional origami pieces. For example, in the collection of 58 traditional pieces in the book *Japanese Origami Encyclopedia* [Yam95], 40 pieces satisfy the condition. Our further investigation revealed that as many as 38 pieces of them can be created by only seven types of folding operations and their compositions. Motivated by this finding, we decided to support the following eight types of operations, i.e., *simple fold*, *tucking*, *sink fold*, *inside reverse fold*, *outside reverse fold*, *pleat fold*, *pig's leg*, and *mirroring*. Note that the names of folding operations are commonly used in the origami community except for "pig's leg" and "mirroring", which we named because they do not have common names.

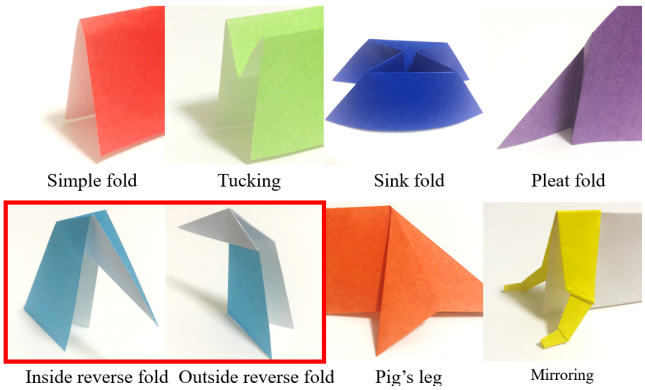



**Figure 2:** *Photos of the folded shapes made by corresponding folding operations. Note that inside/outside reverse folds can be regarded as identical operations, and thus we rename them as "reverse fold" in this paper.*

Figure 2 shows the example shapes obtained by the seven folding operations. We explain each operation as follows. Figures in Section 5.4 show the detailed geometry of each folding operation.

**Simple fold** is the simplest operation to create a single mountain or valley fold along a straight crease line. In our results, this operation is often used to create, e.g., the back of animals.

**Tucking** pushes paper edges inward, often used for adjusting outer shape and beautifying the silhouette

**Sink fold** crushes and pushes a part of the origami shape inward. As a result, the pushed part is inverted and invisible from the outside.

**Inside reverse fold** makes a triangle at a tip of the origami piece by pushing a crease inward. It is often used to make the shapes of bird's beaks or animal's limbs.

**Outside reverse fold** opens a tip of a crease and folds it back. This operation makes a part of the crease covered.

**Pleat fold** creates a step on a face by pushing a part of the face inward.

**Pig's leg** is often used to create the legs of various animals, e.g., pig by lifting up polygon faces.

**Mirroring** is not a folding operation but a duplicating operation for grouped polygons to create mirror-symmetric counterparts (shown in yellow in Figure 2), which is used in complicated models.

Note that, among the eight operations, "inside/outside reverse fold" can be regarded as the same operation by swapping the folded-back part and the remaining part. Hence we regard them as identical operations and rename them as "reverse fold," which reduces the number of internal representations of supported operations from eight to seven. Hereafter we discuss the seven operations.

## 4. User Interface

Our system provides two windows; one is for a 2D view and the other for a 3D view. In the 2D window, the user inputs a reference image if available, draws polygons, and annotates the polygons (Section 4). The user can inspect the resultant origami model in the 3D window. In the initial state, the origami model is represented as a pair of flat front and back polygons. The shape of the model is then updated according to the user-specified annotations. The inputs to our system are the initial polygons of the target origami and annotations to specify folding operations, as explained below.

### 4.1. Initial polygons of target origami

First of all, the user inputs the initial polygons that define the outer shape of the target origami in the flat state by clicking vertex positions (on a reference image if available). The polygon edges represent the paper edges and creases visible from the outside to define geometric shapes of individual polygons. In this step, we do not distinguish whether the user-specified edges are the paper edges or creases. Every time the user draws a polygon, the system rectifies the shape (Section 5.1).

Note that, if a reference image is available, automating this manual step would be possible using combination of image processing and machine learning (e.g., line detection and segmentation), which is out of the scope of this paper.

### 4.2. Annotations for specifying folding operations

The user specifies annotations among the seven candidates corresponding to the folding operations (i.e., simple fold, tucking, sink fold, reverse fold, pleat fold, pig's leg, and mirroring) by clicking and dragging edges or faces. Our system then calculates the folded shape (Section 5.4).

Figure 3 summarizes the folding operations, mouse gestures, annotations supported by our system, and resultant shapes. The annotations in our paper are distinguished by colors, i.e., red for simple fold, green for tucking, blue for sink fold, cyan for reverse fold, magenta for pleat fold, orange for pig's leg, and yellow for mirroring. Note that reverse fold and pig's leg lead to additional simple folds automatically, as shown in the annotations in the fourth and seventh rows. Also note that "reverse fold" and "pleat fold" are specified by the same mouse gesture of a drag between two polygons, but our system recognizes them automatically; if the two polygons share one vertex, our system selects "reverse fold" and otherwise selects "pleat fold". Consequently, the number of annotations that the user should remember is reduced from seven to six.

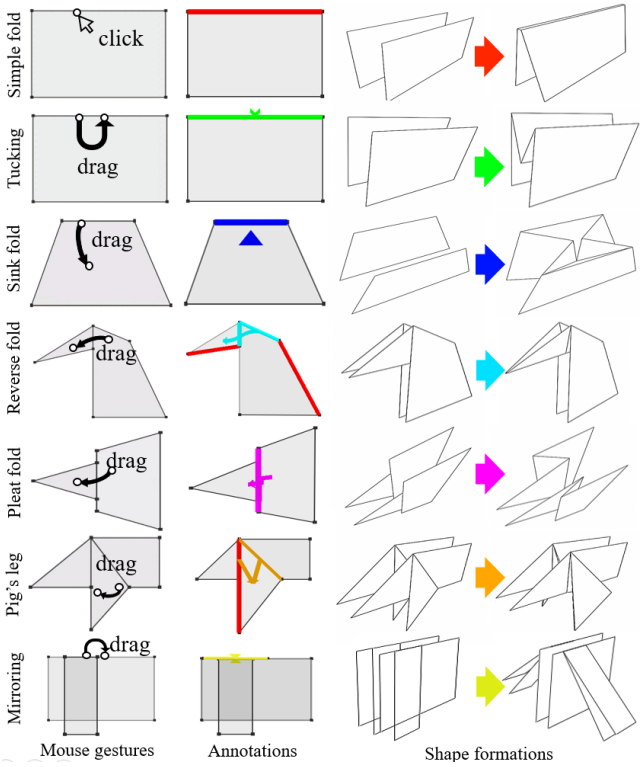



**Figure 3:** *Mouse gestures, annotations, and geometric structures corresponding to folding operations. The annotations are distinguished by colors in our paper. For reverse fold and pig's leg, simple folds are added automatically.*

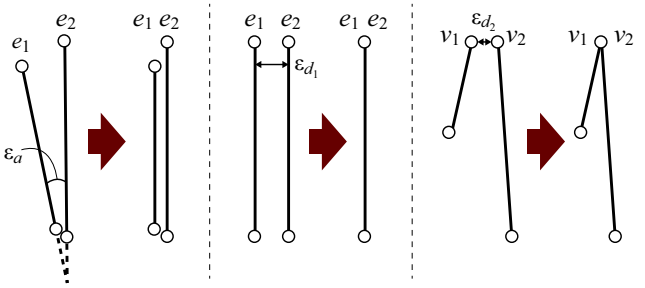
## 5. Generating a 3D Origami

Given the user inputs, our system constructs a 3D origami model as follows. First, every time the user inputs a polygon, our system rectifies the current shape by merging nearby vertices or edges (Section 5.1). Second, every time the user inputs an annotation, our system infers a folding operation and maintains the depth order of polygons as a directed graph (Section 5.2). Meanwhile, our system checks whether the input polygons satisfy the geometric constraints induced by the inferred folding operation, and otherwise rectifies mis-aligned polygons (Section 5.3). Finally, our system integrates the 3D shape structure corresponding to the user-specified folding operation (Section 5.4). Here we explain the details.

### 5.1. Rectification of initial polygons

To rectify possible mis-alignments of the manually-specified initial polygons, our system checks whether each of the following conditions is satisfied in the following order, and modifies vertex positions if satisfied.

**i) Parallel edges (Figure 4, left).** If the angle between edge $e_1$ of the new polygon and edge $e_2$ of an existing polygon is less than threshold $\varepsilon_a$, $e_1$ is adjusted so that $e_1$ becomes parallel to $e_2$. This is accomplished by moving $e_1$'s endpoint (that is farther from $e_2$) along the normal direction of $e_2$.




i) Parallel edges ii) Collinear edges iii) Coinciding vertices

**Figure 4:** *Rectification of initial polygons. The user-specified polygons are modified if the conditions are satisfied.*

**ii) Collinear edges (Figure 4, middle).** If edge $e_1$ of the new polygon and edge $e_2$ of an existing polygon are parallel and the distance between them is less than threshold $\varepsilon_{d_1}$, $e_1$ are moved along the normal direction of $e_2$ so that they become collinear.

**iii) Coinciding vertices (Figure 4, right).** If the distance between vertex $v_1$ of the new polygon and vertex $v_2$ of an existing polygon is less than threshold $\varepsilon_{d_2}$, $v_1$ is moved to the same position as $v_2$.

In our system, we set $\varepsilon_a = 10^\circ$ and $\varepsilon_{d_1} = \varepsilon_{d_2} = 10$ pixels for a screen of $800 \times 600$ pixels. Note that condition i) is applied only to nearby edges; we apply condition i) if $0.2 \min(|e_1|,|e_2|) > dist(e_2, v_1)$ and $0.2 \min(|e_1|,|e_2|) > dist(e_2, v_2)$, where $v_1$ and $v_2$ are the closer endpoints of $e_1$ and $e_2$ respectively, and $|e|$ denotes the length of edge $e$ and $dist(e_2, v)$ the distance between $e_2$ and $v$. We just check and apply the conditions sequentially, and thus any race condition does not occur. Additionally, when the user employs mirroring, polygons belonging to the same mirrored group are only rectified among themselves because each group should be duplicated independently.

### 5.2. Depth order and thickness representation

To determine the 3D origami shape with thickness, we must know which polygon should be in front of or behind other adjacent polygons. Such depth order of polygons must be maintained consistently against the sequence of user-specified folding operations. Our system maintains the depth order as a directed graph where each node corresponds to a polygon and each directed edge represents that a polygon is in front of the other adjacent polygon. Note that directed edges are assigned only among adjacent polygons, and thus the relationships are local. We do not handle folding operations that cause cyclic relationships, e.g., *twist fold*, and thus the graph consists of one or more trees whose root node(s) lie(s) on top of other polygons. In the beginning, each node is independent, and there are no directed edges. The directed graph is updated every time the user specifies an annotation.

For explanation, we define a right-handed coordinate system where the screen corresponds to the $xy$ plane whose origin lies in the lower-left corner of the screen, and the positive $z$ direction corresponds to the direction from the screen to the viewer. The symmetry plane of the front-and-back symmetry corresponds to the $xy$ plane (i.e., the screen). Hereafter we only consider the polygons with positive $z$ coordinates because of the front-and-back symmetry. If polygon $F_a$ is located above polygon $F_b$, i.e., $F_a$ has a larger $z$ coordinate than $F_b$, this relationship is denoted as a directed edge $F_a \to F_b$.

Among the seven operations supported by our system, reverse fold, pleat fold, and pig's leg determine the local depth order. On the other hand, simple fold, tucking, and sink fold do not change the depth order because these operations only connect polygons with their mirror-symmetric counterparts. As an exception, only mirroring creates a new sub-graph whose elements are the polygons in the polygon group. The new sub-graph does not affect the ordering of other nodes in the original graph. If the user annotates one of the three operations, our system adds one or two directed edges. Specifically, if the user specifies reverse fold or pleat fold by dragging from polygon $F_a$ to polygon $F_b$, our system assigns a new depth order $F_a \to F_b$. If the user annotates pig's leg, three polygons are involved (Figure 3, bottom row). Let $F_a$ be the polygon where the mouse-drag trajectory lies, $F_s$ and $F_e$ be the polygons close to the starting and ending points of the trajectory, respectively. For pig's leg, our system assigns two relationships $F_a \to F_s$ and $F_a \to F_e$.

We explain how the directed graph is updated with an example shown in Figure 5. In this example, the four annotations (Figure 5(a)) determine the depth order; three reverse folds between (1) $F_a$ and $F_b$, (2) $F_b$ and $F_c$, and (3) $F_e$ and $F_f$, as well as (4) pig's leg among $F_c$, $F_d$, and $F_e$. Figure 5(b) shows the step-by-step illustrations how the directed graph is updated according to the annotations (1)-(4) in Figure 5(a). With three reverse folds (1)-(3), directed edges (1) $F_b \to F_a$, (2) $F_c \to F_b$, and (3) $F_e \to F_f$ are added, respectively. Similarly, with pig's leg (4), directed edges $F_d \to F_c$ and $F_d \to F_e$ are added. As is apparent from this example, the final graph structure does not depend on the order in which annotations are specified.

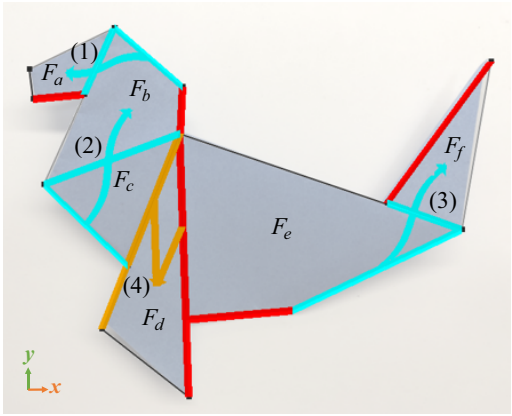
Now we explain how to assign depth offsets to layered polygons in order to represent thickness due to interlayer gaps. Let $d_i$ be the distance from tree's root node to node $i$ (i.e., polygon $i$, where $i \in \{1, 2, \ldots, N\}$ and $N$ is the number of nodes), and $d_{max}$ be the height of the tree defined as $d_{max} = \max_i d_i$. Our system calculates the $z$ coordinate $z_i$ of polygon $i$ as follows:

$$z_i = \Delta d \, (d_{max} - d_i + 1), \tag{1}$$

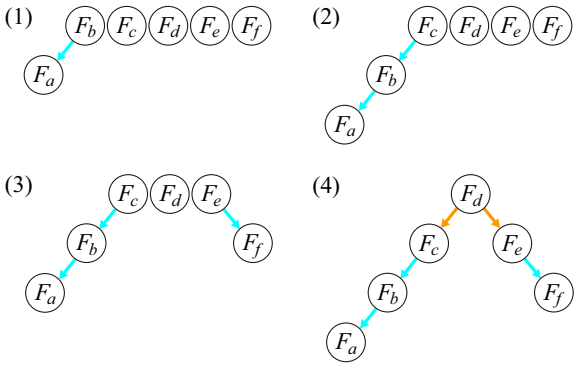
where $\Delta d$ is the user-specified coefficient that defines the unit gap width between successive layers. For example, if we set $\Delta d = 1$, $\{z_i\}$ of polygons from $F_a$ to $F_f$ in Figure 5(a) become 1, 2, 3, 4, 3, and 2, respectively. After calculating $z_i$, our system translates polygon $i$ along the $z$ direction, and applies further modifications corresponding to folding operations, as described in the next subsections. When mirroring is specified, a polygon group is translated so that it is attached to another group and then rotated around the attached edge, as described in Section 5.4.7

### 5.3. Constraint-based polygon rectification

After a folding operation is specified, our system checks whether the manually-specified polygons satisfy the geometric constraint(s) induced by the folding operation. If not, our system rectifies the vertex positions of the input polygons automatically. Our system supports such rectifications for reverse fold and pleat fold.




(a) Polygons with annotations (1) - (4)




(b) Step-by-step illustrations of updated depth-order graphs

**Figure 5:** *Construction of a directed graph of the depth order of polygons. (a) The user-specified annotations (1) - (4) sequentially update (b) the graphs accordingly. The specified order of annotations does not affect the resultant depth order.*

The left image of Figure 6(a) illustrates the geometric constraint of reverse fold. Triangles $\triangle v_2v_4v_5$ and $\triangle v_1'v_4v_5$ show the ideal configuration before and after reverse fold, respectively; they should be symmetric w.r.t. crease line $v_4v_5$. However, the user-specified triangle $\triangle v_1v_3v_5$ (blue) is not aligned to the ideal triangle $\triangle v_1'v_4v_5$ in this case. Our system thus moves vertex $v_1$ to $v_1'$ and vertex $v_3$ to $v_3'$ after the annotation is specified. If the shape before folding is an $n$-gon $(n > 3)$ (Figure 6(b)), our system moves $v_3$ and $v_5$ so that they are aligned with the ideal mirror image, while keeping the lengths of the line segments connected to $v_3$ and $v_5$.

In pleat fold (Figure 6(c)), the angle between $e_3$ and $e_4$ should be equal to the angle between $e_1$ and $e_2$ because the shape before folding is a single triangle. However, the user-specified triangle consisting of $e_3$ and $e_4$ does not satisfy this constraint in this case. Under the assumption that $e_1$ and $e_3$ as well as $e_2$ and $e_4$ often become parallel in real origami pieces, our system fixes $v_2$ and $v_3$, and translates vertex $v_1$ to $v_1'$ so that the parallel-edge assumption is satisfied, after the annotation is specified. If the shape before folding is an $n$-gon $(n > 3)$ (Figure 6(d)), our system also fixes $v_2$ and $v_3$, and moves the endpoints of $e_3$ and $e_4$ other than $v_2$ and $v_3$ so that

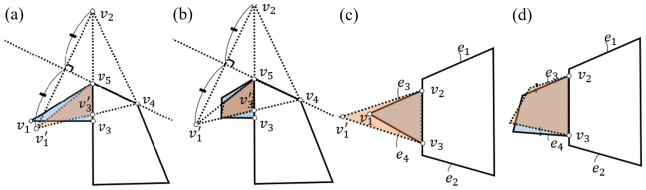



**Figure 6:** *Constraint-based polygon rectification for reverse fold ((a) and (b)) and pleat fold ((c) and (d)). The user-specified polygons (blue) are modified (red) to satisfy the geometric constraints.*

$e_3$ and $e_4$ become parallel to $e_1$ and $e_2$, respectively, while keeping the lengths of $e_3$ and $e_4$.

### 5.4. Shape construction based on annotations

After calculating the initial depth $z_i$ (Equation (1)) for each polygon $i$, our system modifies polygon shapes according to the user-specified annotations by moving vertex positions and completing gaps with new polygons. As shown in the rightmost column in Figure 3, a pair of mirrored shapes are transformed according to each of the seven operations, as described below. Note that we explain pleat fold and pig's leg for polygons only on the positive $z$ side because they are symmetric w.r.t. the $xy$ plane.

#### 5.4.1. Simple fold

For a simple fold, we must satisfy the constraint that the mirrored pair of front and back shapes must be connected via a crease line. To express a fold line and interlayer gap, our system inserts an elongated polygon along the crease to connect each mirrored pair. Let $e$ be the clicked edge, $e'$ be the edge symmetrical to $e$, $\{v_1, v_2\}$ and $\{v_1', v_2'\}$ be the pairs of the endpoints of $e$ and $e'$, respectively. From the unconnected initial state, our system modifies the shape as follows (Figure 7).

**Step 1:** Bring edges $e$ and $e'$ closer to each other by multiplying their $z$ coordinates with coefficient $\alpha \in (0, 1)$. $\alpha$ is set as 0.1 by default, and can be changed by the user.

**Step 2:** Add a thin quad $\square v_1v_2v_2'v_1'$ to connect edges $e$ and $e'$.

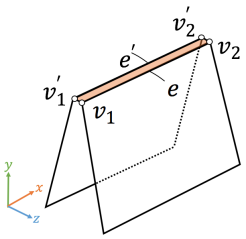



**Figure 7:** *Structure of simple fold. The pair of front and back polygons are connected with a thin quad.*

#### 5.4.2. Tucking

Tucking pushes polygons inward, and thus we must determine the tucked depth, which cannot be observed from the outside. Assuming that the accurate depth of tucking is not required to generate

plausible outer shapes, we determine the tucked depth empirically. Let $e$ be the edge where the dragging starts, $e_{prev}$ and $e_{next}$ be the edges previous and next to $e$ clockwise around the face adjacent to $e$. We also define $e'$, $\{v_1, v_2\}$ and $\{v'_1, v'_2\}$ similarly to the case of a simple fold. Our system calculates the tucked shape as follows (Figure 8).

**Step 1:** Generate two vertices $v_a$ and $v_b$. $v_a$ and $v_b$ are located on the $xy$ plane along the projections of two half-lines emanating from $v_1$ and $v_2$ along $e_{prev}$ and $e_{next}$, respectively (Figure 8, left). The distance from the plane $\mathbf{v}_1\mathbf{v}_2\mathbf{v}'_2\mathbf{v}'_1$ to vertex $v_a$ (or $v_b$) is determined by the tucked depth $d_{tucking}$. In our system, we set $d_{tucking} = 0.25\,|e_{prev}|$ (Figure 8, right).

**Step 2:** Add two quads $\square v_1 v_2 v_b v_a$ and $\square v'_1 v'_2 v_b v_a$.

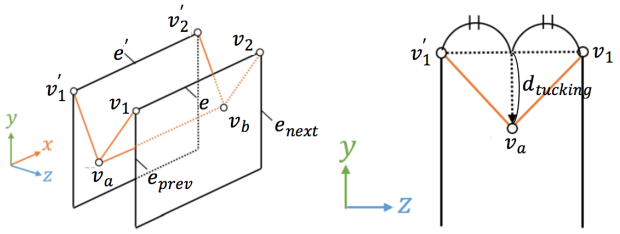



**Figure 8:** *Structure of tucking. The pair of front and back polygons are connected with "V"-shaped polygons.*

### 5.4.3. Sink fold

Similarly to tucking, the depths of sink folds cannot be observed from the outside, and thus we determine the depths empirically. In addition to the definitions above, let $v_4$ and $v_3$ be the endpoints of $e_{prev}$ and $e_{next}$ other than $v_1$ and $v_2$, respectively. We also define $v'_3$ and $v'_4$ as the mirrored counterparts of $v_3$ and $v_4$. The structure of sink fold is then calculated as follows (Figure 9).

**Step 1:** Generate five vertices $v_a$, $v_b$, $v_c$, $v_d$, and $v_e$. $v_a$ and $v_b$ are located along a line parallel to $e$ such that their midpoint is the centroid of $\{v_1, v_2, v'_1, v'_2\}$. The distance between $v_a$ and $v_b$ is determined by $d_{sink}$. In our system, we set $d_{sink} = 0.1\,|e|$. $v_c$ and $v_d$ are determined similarly with the centroid of $\{v_3, v_4, v'_3, v'_4\}$. $v_e$ is the centroid of $\{v_a, v_b, v_c, v_d\}$.

**Step 2:** Add six triangles $\triangle v_1 v_e v_2$, $\triangle v_1 v_a v_e$, $\triangle v_2 v_e v_b$, $\triangle v'_1 v_e v'_2$, $\triangle v'_1 v_a v_e$, and $\triangle v'_2 v_e v_b$, and four quads $\square v_1 v_4 v_c v_a$, $\square v_2 v_b v_d v_3$, $\square v'_1 v'_4 v_c v_a$, and $\square v'_2 v_b v_d v'_3$ (Figure 9, left). The right image of Figure 9 illustrates the edge connections.

### 5.4.4. Reverse fold

In reverse fold, faces are also pushed inward but the pushed depth is constrained, unlike tucking and sink fold. Let $F_1$ and $F_2$ be the polygons where the mouse drag starts and ends. Suppose that $F_1$ and $F_2$ are already assigned different $z$ coordinates (Section 5.2). On polygon $F_2$, let $e_2$ be the edge where $F_1$ and $F_2$ touch each other, $\{v_2, v_4\}$ be $e_2$'s endpoints at the tip and the other side, and $e_4$ be another edge connected to $v_4$. On polygon $F_1$, let $v_1$ be the vertex that has the same $x, y$ coordinates as $v_2$, $e_1$ be the edge connected to $v_1$ along the outline of the whole shape, and $v_3$ be the other endpoint of $e_1$. Reverse fold is then realized as follows (Figure 10).

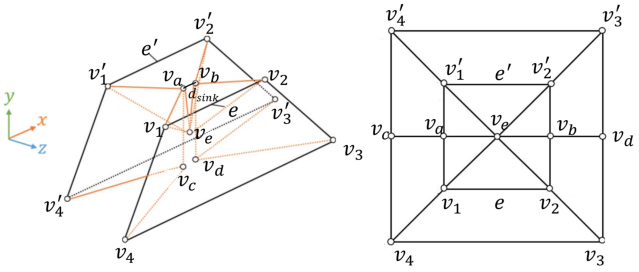



**Figure 9:** *Structure of sink fold. The right image illustrates the edge connections.*

**Step 1:** Merge $v_1$ and $v_2$ by choosing either one having the larger $z$ coordinate. Hereafter we assume $v_1$ is chosen.

**Step 2:** Add triangle $\triangle v_1 v_3 v_4$.

**Step 3:** Apply the operations of simple fold to both $e_3$ and $e_4$.

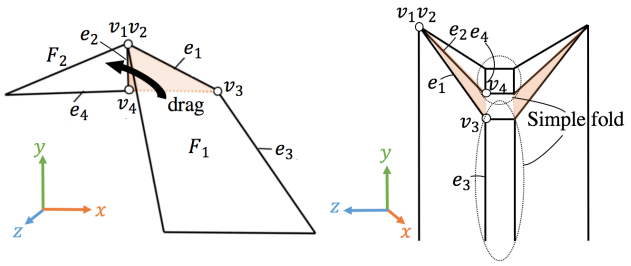
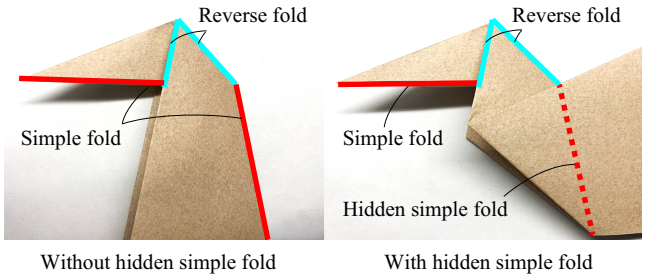



**Figure 10:** *Structure of reverse fold. The right image shows a side view from positive x direction.*

Note that in rare cases the edge that simple folds should be applied to (i,e., edge $e_3$) might be hidden (Figure 11), where the above procedure does not work. To handle this problem, we skip Step 3 for such a hidden edge. Note that, even if Step 3 is omitted, the paired polygons that form the bird's head are not disconnected thanks to the simple fold at its beak. The user can also specify more folding operations if necessary.




Without hidden simple fold With hidden simple fold

**Figure 11:** *Problematic case of reverse fold. If the edge to which simple fold should be applied is hidden (right), we ignore Step 3 for such an edge.*

#### 5.4.5. Pleat fold

Pleat fold generates polygons having different $z$ coordinates from a single polygon. Suppose that polygons $F_1$ and $F_2$ are defined similarly to reverse fold, and assigned the relationship $F_1 \rightarrow F_2$ and corresponding $z$ coordinates (Section 5.2). Let $e_1$ and $e_2$ be the edges on $F_1$ and $F_2$ where $F_1$ and $F_2$ touch each other, respectively. On polygon $F_2$, let $e_2'$ and $e_2''$ be the edges that share endpoints with $e_2$. Pleat fold is realized as follows (Figure 12).

**Step 1:** Generate two vertices $v_a$ and $v_b$. $v_a$ and $v_b$ are on the lines extended from $e_2'$ and $e_2''$, and are determined so that segment $v_a v_b$ and $e_2$ become parallel and the distance between them be $d_{pleat}$ (Figure 12, right). In our system, we set $d_{pleat}$ as half the length of the shorter edge of the two edges that share $e_1$'s endpoints.

**Step 2:** Add the hidden face adjacent to $e_1$ (shown in orange in Figure 12, left).

**Step 3:** Enlarge $F_2$ along $e_2'$ and $e_2''$ so that $F_2$ touches segment $v_a v_b$.

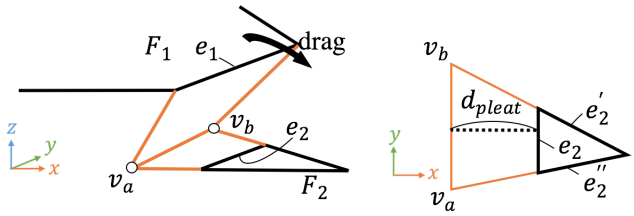



**Figure 12:** *Structure of pleat fold. One polygon is added and $F_2$ is enlarged to connect $F_1$ and $F_2$.*

#### 5.4.6. Pig's leg

Pig's leg generates leg-like shapes by lifting polygon structures. Let $F_1$ be the polygon where the mouse-drag trajectory lies, and $F_2$ and $F_3$ be $F_1$'s adjacent polygons close to the starting and ending points of the trajectory, respectively. Suppose that polygons $F_1$, $F_2$, and $F_3$ are assigned different $z$ coordinates according to the relationships $F_1 \rightarrow F_2$ and $F_1 \rightarrow F_3$ (Section 5.2). We further define edges $e_1$, $e_2$, and $e_3$ on $F_1$, $F_2$, and $F_3$, respectively; $e_1$ and $e_2$ are the edges close to the starting point of the trajectory while $e_3$ is close to the ending point. $e_1$, $e_2$, and $e_3$ have vertices whose $x, y$ coordinates are the same but $z$ coordinates are different. Similarly to reverse fold, we merge the three vertices by choosing either one having the largest $z$ coordinate and denote the chosen one as $v$. Let $e_1'$ and $e_2'$ be the edges on $F_1$ and $F_2$ that share endpoints with $e_1$ and $e_2$ but do not share $v$, respectively. Pig's leg is realized as follows (Figure 13).

**Step 1:** Merge the three vertices of $e_1$, $e_2$, and $e_3$ that share the same $x, y$ coordinates to generate $v$, as explained above.

**Step 2:** Generate two vertices $v_a$ and $v_b$. $v_a$ is located where $e_2'$ is extended toward $F_1$ by distance $d_{pig_1}$ while $v_b$ is above $v_a$ by distance $d_{pig_2}$ toward positive $z$ direction. In our system, we set $d_{pig_1}$ as half the distance between two endpoints of $e_2$ and $e_3$ other than $v$, and $d_{pig_2}$ as half the distance between $F_1$ and $F_3$.

**Step 3:** Add three triangles consisting of i) $v_a$ and $e_1$'s both endpoints, ii) $v_b$ and $e_3$'s both endpoints, and iii) $v_b$ and both endpoints of the edge connected to $v$ other than $e_1$ on $F_1$.

**Step 4:** Enlarge $F_2$ along $e_2'$ so that $F_2$ touches $v_a$.

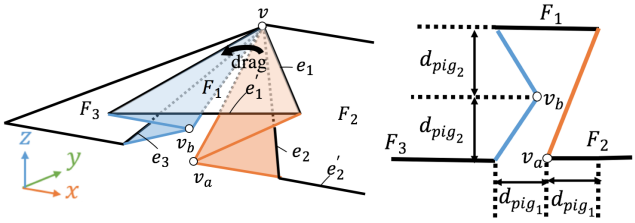



**Figure 13:** *Structure of pig's leg. The right image shows the view from negative $y$ direction.*

#### 5.4.7. Mirroring

In mirroring, polygon groups are specified by the user, each group is attached to another group at a single edge, and the attached group is duplicated to create its mirror-symmetric counterpart. Polygon grouping is enabled by explicitly specifying the grouping mode in our system. Attaching a group to another is specified by a mouse drag (Figure 3, bottom). Let $e_1$ and $e_2$ be the edges where the dragging starts and ends, and $e_1'$ be the mirror-symmetric counterpart of $e_1$. We also define $e_1''$ as an edge at the middle of $e_1$ and $e_1'$, which is always on the $xy$ plane. Mirroring is realized as follows (Figure 14).

**Step 1:** Merge $e_1$ and $e_1'$ into $e_1''$ (Figure 14(1)).

**Step 2:** Translate the merged edge $e_1''$ to $e_2$ while keeping the shape of the polygon group (blue) (Figure 14(2)).

**Step 3:** Rotate the polygon group around edge $e_2$ by angle $a_{mirror}$ (Figure 14(3)). $a_{mirror}$ is set as 15° by default, and can be changed interactively by the user.

**Step 4:** Duplicate the polygon group to create a mirror-symmetric counterpart (Figure 14(4)).

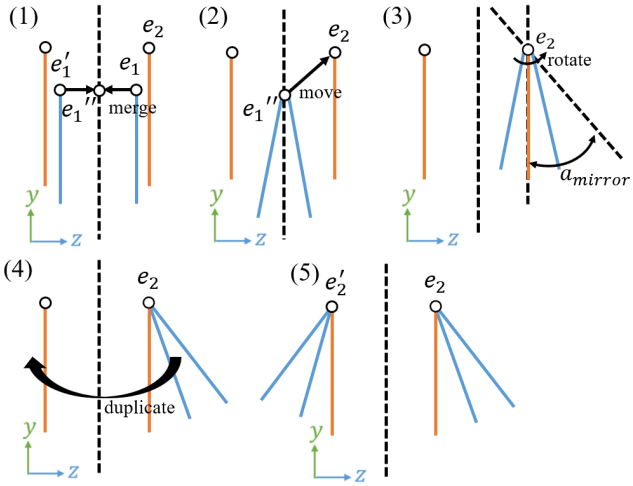



**Figure 14:** *Construction process of mirroring (viewed from $+x$ direction). A polygon group (blue) is moved and duplicated.*

## 6. Results

Our prototype system was implemented in C++ with OpenGL and Qt libraries and run on an off-the-shelf PC. The screen sizes for 2D- and 3D-view windows are both $800 \times 600$ pixels. If the reference image is larger than the screen size of the 2D-view window, the image is fitted to the screen by uniform scaling. Among the parameters in our system, $\Delta d$ (Section 5.2) and $a_{mirror}$ (Section 5.4.7) are the most important to determine the overall outer shape. Whereas the user can change any parameters interactively, we only changed $\Delta d$ and $a_{mirror}$ and kept other parameters fixed in all of our results. Please refer to our accompanying movie for captured demo videos and a detailed comparison with the origami model created using Maya.

The origami pieces in our results are traditional artworks selected from the book [Yam95]. We used reference images for all the results. Our origami models were created using our system, but the images in our results were rendered with textures using the standard 3D software, *Autodesk Maya* [Aut]. Table 1 summarizes the statistics of the origami models created using our system. The types of annotations used in our results are visualized by the colors illustrated in Figure 3. Figure 15 shows seven examples of animal origami models created using our system.

**Table 1:** *Statistics of our origami models. #Folds denotes the numbers of folding operations required when we create corresponding physical origami pieces, #Anno. the numbers of annotations, #Poly the numbers of polygons (generated initially and finally), respectively.*

| Origami models | #Folds | #Anno. | #Poly | |
|---|---|---|---|---|
| | | | Initial | Final |
| Fur seal (Fig. 1) | 14 | 4 | 6 | 28 |
| Knot (Fig. 15) | 11 | 2 | 5 | 24 |
| Elephant (Fig. 15) | 26 | 10 | 6 | 34 |
| Dragon (Fig. 15) | 142 | 25 | 15 | 98 |
| Snail (Fig. 15) | 17 | 7 | 5 | 33 |
| Tiger (Fig. 15) | 69 | 15 | 10 | 71 |
| Crow (Fig. 15) | 24 | 9 | 7 | 44 |
| Fowl (Fig. 15) | 18 | 4 | 7 | 29 |
| Mandarin duck (Fig. 16) | 20 | 4 | 4 | 18 |
| Pig (Fig. 16) | 15 | 5 | 6 | 30 |
| Bird (Fig. 16) | 16 | 5 | 5 | 22 |
| Tangram (Fig. 18) | (n/a) | 5 | 5 | 24 |
| Tangram (Fig. 18) | (n/a) | 9 | 7 | 32 |
| Penguin (Fig. 20) | 17 | 3 | 5 | 24 |
| Carp (Fig. 20) | 12 | 5 | 4 | 20 |
| Knot (Fig. 20) | 19 | 7 | 6 | 27 |

### 6.1. Comparison with conventional modeling software

To evaluate the plausibility of the origami models created using our system, we compared them with models with accurate inner structures created manually using Maya, and physical origami pieces. Figure 16 shows the resultant origami models (i.e., "Mandarin duck", "Pig", and "Bird") while Figure 17 shows the corresponding *crease patterns*, i.e., the unfolded results of the origami models.

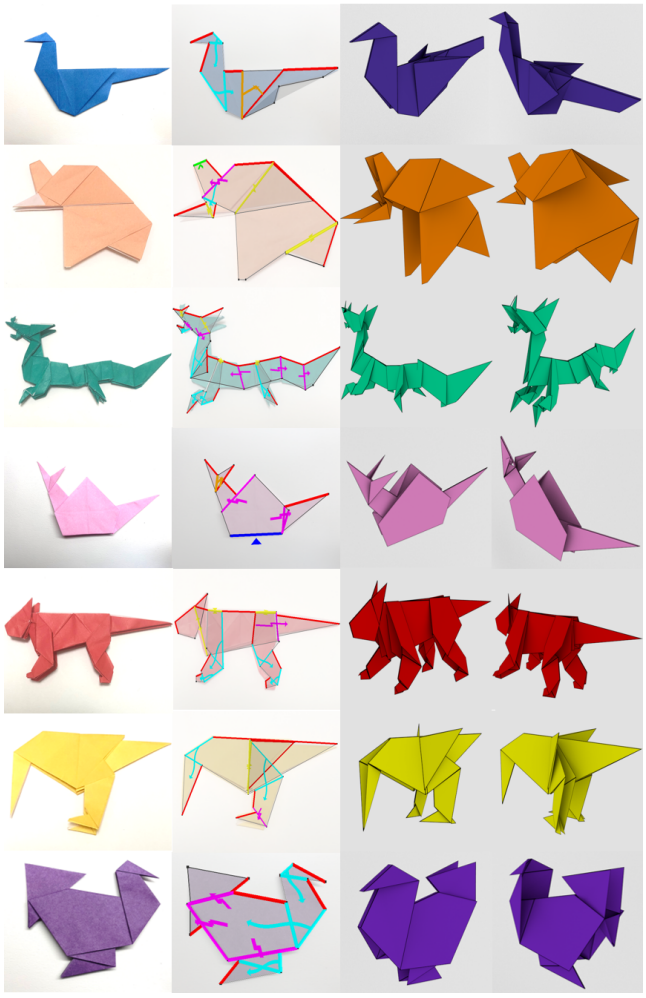


**Figure 15:** *Examples of origami animals created using our system.*

The color in the crease patterns indicates the visibility of each face; from blue to red, fully visible from outside, slightly hidden, largely hidden, and fully hidden. As we can see from Figure 16, the outer shapes of our origami models are similar to those created using Maya and real origami pieces. On the other hand, from Figure 17, we can see that our origami models consist of fewer polygons. Regarding the complexity of the inner structures, the maximum numbers of stacked layers are $\{12, 12, 16\}$ for the Maya models while $\{6, 6, 6\}$ for our models. The tree heights $d_{max}$ (Section 5.2) of our models are $\{3, 3, 3\}$. These models were created by one of the authors, and the time spent for creating each model is one or two minutes using our system and more than one hour using Maya.

### 6.2. Modeling of pseudo-origami model

We demonstrate the wide applicability of our system by modeling pseudo-origami models. Figure 18 shows that plausible origami-like models can be created from the image of tangram puzzles and Figure 19 shows the corresponding *unfolded pattern* for Figure 18.

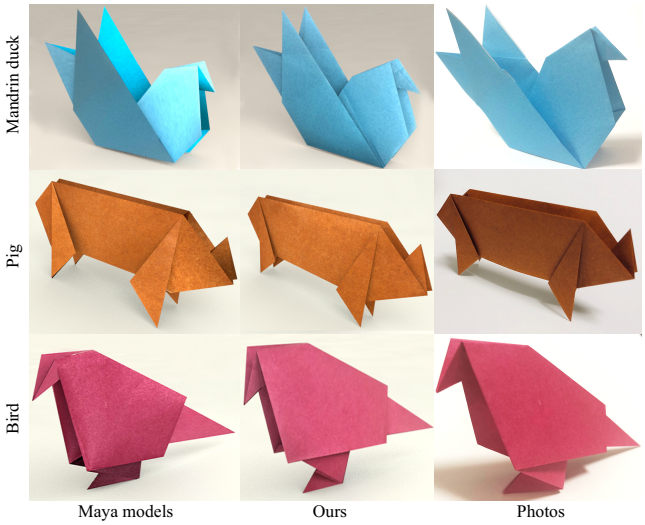



**Figure 16:** *Comparisons with models created using Maya and our system as well as photos of physical origami pieces. The models by Maya have accurate inner structures. Ours have outer shapes similar to the counterparts.*

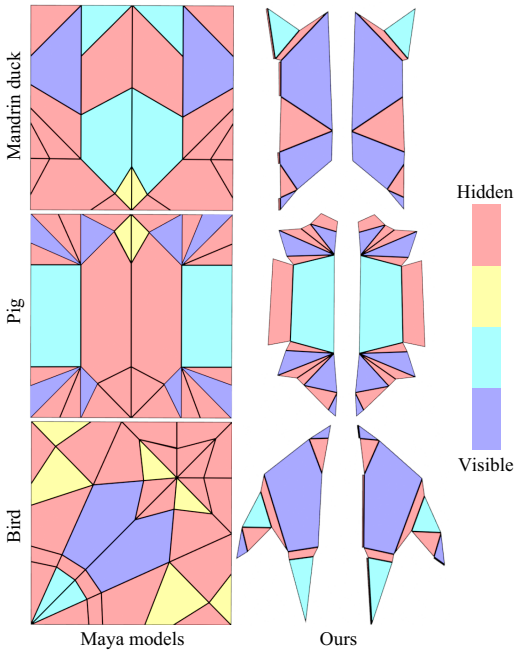



**Figure 17:** *Crease patterns (i.e., unfolded models) with colors encoding visibility for models created using Maya and our system. Our results have fewer polygons and most polygons are visible from outside.*

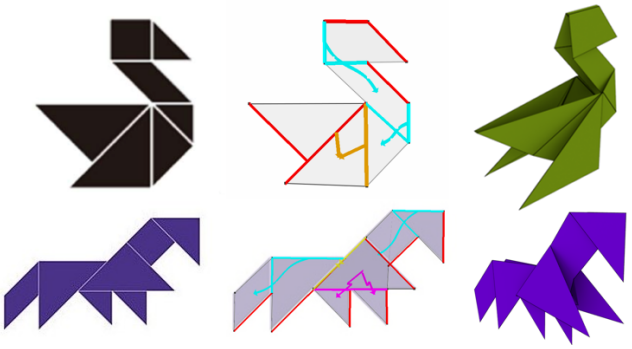


**Figure 18:** *Pseudo-origami models created from reference images of tangram puzzles.*

Note that these models cannot be realized by folding a sheet of paper because it consists of un-developable surfaces; the sum of face angles exceeds $2\pi$ at some vertices. To fabricate physical models, several polygons must be separated as shown in Figure 19.

### 6.3. User studies

**Evaluation of the usability of our system.** We conducted a user study where seven subjects were requested to create origami models using our system. Four of them were familiar with origami while the others have experience of origami only in their childhood. After ten-minute tutorials, the subjects were first requested to create physical origami models in the real world, and then started to model "Penguin", "Carp", and "Knot" until they got satisfied. Figure 20 shows typothe resulting models created by one of the subjects. Table 2 summarizes the statistics of the times required for creating the models. Drawing initial polygons does not seem laborsome because all the subjects finished this task within one minute and no subjects reported difficulty. Also, all subjects could recognize what types of annotations should be specified, which indicates that the tutorial and preliminary physical origami modeling were sufficient even for novice users to exploit our system. We had expected that the required time would depend on the knowledge of folding operations. Surprisingly, however, the average time was one minute, and we could not observe large differences between the familiar and the novice.

**Comparison with a generic system for origami modeling.** We also requested two subjects to use a generic modeling system [FKMF07] to create the models same as Figure 20. The system of [FKMF07] is designed to faithfully trace the manual folding process for modeling general origami models. The subjects suffered from self-intersections during modeling (e.g., Figure 21), and took about 15 minutes on average for each model, which is about 7.5 times longer than using our system.

## 7. Discussion

Here we summarize the limitations and expressivity of our system. Due to our assumptions, our system does not support shapes that are not mirror-symmetric or have trees with cyclic depth orders. If the user tries to create models with cyclic depths, our system will output separate trees whose depths are all one. Also, our system currently does not have a mechanism to avoid self-collisions. Fortunately, self-collisions are not noticeable in most of our results.

**Table 2:** *Times (in seconds) required for inputing polygons and annotations in our user study with seven subjects.*

| Models | Polygons | | | Annotations | | |
|---|---|---|---|---|---|---|
| | Average | Std.dev. | Median | Average | Std.dev. | Median |
| Penguin | 36.3 | 8.8 | 40.0 | 32.4 | 11.6 | 30.0 |
| Carp | 29.1 | 9.4 | 25.0 | 46.0 | 7.2 | 43.0 |
| Knot | 38.3 | 12.3 | 38.0 | 49.4 | 11.6 | 50.0 |

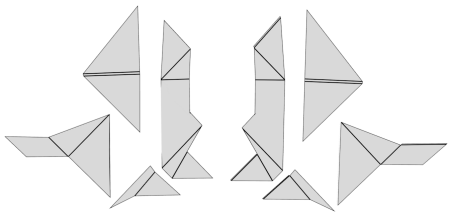


**Figure 19:** *Unfolded pattern of the pseudo-origami model shown in Figure 18, bottom.*

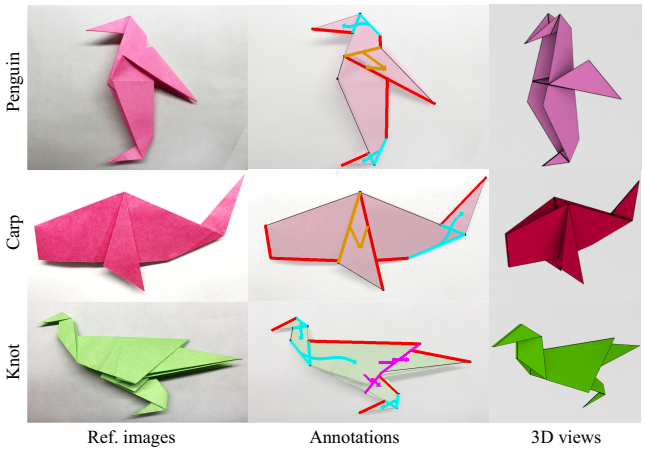



**Figure 20:** *Origami models created in our user study.*

Although our system only supports the seven types of folding operations and a duplicating operation, the expressivity is larger than expected; by emulating non-supported operations with supported ones, we can still create plausible shapes (e.g., the legs of the Tiger model in Figure 15 and the accompanying video). Moreover, as demonstrated in Section 6.2, our system can also handle shapes that are not physically realizable.

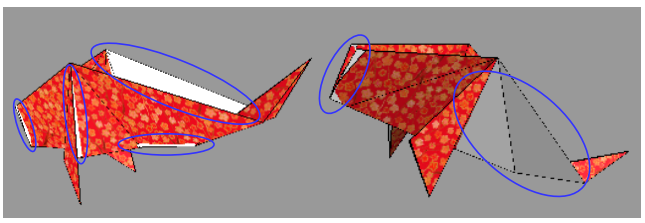


**Figure 21:** *Carp model created using the system of [FKMF07]. The blue ovals indicate self-intersections.*

## 8. Conclusion

In this paper, we have proposed an efficient system for modeling flat origami pieces in a single view, while omitting accurate inner structure yet accounting for the thickness due to interlayer gaps. The thickness representation is realized by managing the depth order of polygons as a directed graph and adjusting the initial $z$ coordinates of polygons with reference to the graph. Our system automatically rectifies the user inputs and integrates the shape structures corresponding to the user-specified folding operations. The user can create plausible 3D origami models quickly by specifying only visible polygons in the flat state as well as the seven types of annotations using simple mouse gestures. Although we focused on flat origami having a front-and-back symmetry, our system can handle most of the animal pieces in a traditional origami book [Yam95]. Our user study revealed that even novice users without the specialized knowledge and experience on origami and 3D modeling can create plausible origami models within one or two minutes for each.

For future work, our system can be extended in several ways. Although we did not implement detection of self-intersections or discrimination of physical realizability, these features will enhance the usability of the system. Supporting additional folding operations and geometric structures is another direction. For example, a variety of 3D origami pieces can be designed by supporting a balloon-like 3D shape, which can be found in one of the most famous origami "crane". If a reference image is available, automatic recognition of input polygons would be quite beneficial to the user. Supporting origami models having a cyclic depth order would also be an interesting direction for future work.

## References

[Aut] AUTODESK I.: Maya. https://www.autodesk.com/products/maya/overview. 9

[Bat07] BATEMAN A.: Tess: origami tessellation software. http://www.papermosaics.co.uk/software.html, 2007. 3

[BCV*15] BESSMELTSEV M., CHANG W., VINING N., SHEFFER A., SINGH K.: Modeling character canvases from cartoon drawings. *ACM Trans. Graph. 34*, 5 (2015), 162:1–162:16. 2

[EBC*15] ENTEM E., BARTHE L., CANI M., CORDIER F., VAN DE PANNE M.: Modeling 3D animals from a side-view sketch. *Computers & Graphics 46* (2015), 221–230. 2

[EPB*19] ENTEM E., PARAKKAT A., BARTHE L., MUTHUGANAPATHY R., CANI M.-P.: Automatic structuring of organic shapes from a single drawing. *Computers Graphics 81* (03 2019). 2

[FKMF07] FURUTA Y., KIMOTO H., MITANI J., FUKUI Y.: Computer model and mouse interface for interactive virtual origami operation (in japanese). *IPSJ Journal 48*, 12 (dec 2007), 3658–3669. 3, 10, 11

[GIZ09] GINGOLD Y. I., IGARASHI T., ZORIN D.: Structured annotations for 2D-to-3D modeling. *ACM Trans. Graph. 28*, 5 (2009), 148:1–148:9. 2

[GP94] GUEST S. D., PELLEGRINO S.: The Folding of Triangulated Cylinders, Part I: Geometric Considerations. *Journal of Applied Mechanics 61*, 4 (1994), 773. 3

[IM10] IGARASHI T., MITANI J.: Apparent layer operations for the manipulation of deformable objects. *ACM Trans. Graph. 29*, 4 (2010), 110:1–110:7. 2

[IMKG09] IDA T., MARIN H., KASEM M., GHOURABI F.: Computational origami system eosin. *Origami4 Fourth International Meeting of Origami* (2009), 285–293. 3

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: A sketching interface for 3D freeform design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1999* (1999), pp. 409–416. 2

[KFC*08] KILIAN M., FLÖRY S., CHEN Z., MITRA N. J., SHEFFER A., POTTMANN H.: Curved folding. *ACM Trans. Graph. 27*, 3 (2008), 75:1–75:9. 3

[Lan11] LANG R. J.: *Origami Design Secrets, 2nd edition*. A.K Peters/CRC Press, 2011. 3

[MEKM17] MIYAMOTO E., ENDO Y., KANAMORI Y., MITANI J.: Semi-automatic conversion of 3D shape into flat-foldable polygonal model. *Comput. Graph. Forum 36* (2017), 41–50. 3

[Mit05] MITANI J.: ORIPA: Origami Pattern Editor. http://mitani.cs.tsukuba.ac.jp/oripa/, 2005. 3

[MYYT96] MIYAZAKI S., YASUDA T., YOKOI S., TORIWAKI J.: An origami playing simulator in the virtual space. *Journal of Visualization and Computer Animation 7*, 1 (1996), 25–42. 2, 3

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: FiberMesh: designing freeform surfaces with 3D curves. *ACM Trans. Graph. 26*, 3 (2007), 41. 2

[NPO13] NARAIN R., PFAFF T., O'BRIEN J. F.: Folding and crumpling adaptive sheets. *ACM Trans. Graph. 32*, 4 (2013), 51:1–51:8. 2

[PDRK14] PACZKOWSKI P., DORSEY J., RUSHMEIER H. E., KIM M. H.: Paper3D: bringing casual 3D modeling to a multi-touch interface. In *The 27th Annual ACM Symposium on User Interface Software and Technology, UIST '14* (2014), pp. 23–32. 2

[RSW*07] ROSE K., SHEFFER A., WITHER J., CANI M., THIBERT B.: Developable surfaces from arbitrary sketched boundaries. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing* (2007), pp. 163–172. 2

[SRH*15] SCHRECK C., ROHMER D., HAHMANN S., CANI M., JIN S., WANG C. C. L., BLOCH J.: Nonsmooth developable geometry for interactively animating paper crumpling. *ACM Trans. Graph. 35*, 1 (2015), 10:1–10:18. 2

[Sta12] STACHEL H.: A flexible planar tessellation with a flexion tiling a cylinder of revolution. *Journal for Geometry and Graphics 16*, 2 (2012), 153–170. 3

[TAC10] TACHI T.: Origamizing polyhedral surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 2 (2010), 298–311. 3

[TBWP16] TANG C., BO P., WALLNER J., POTTMANN H.: Interactive design of developable surfaces. *ACM Trans. Graph. 35*, 2 (2016), 12:1–12:12. 2

[Yam95] YAMAGUCHI M.: *Japanese Origami Encyclopedia (in Japanese)*. Natsumesha, 1995. 3, 9, 11

[YKD11] YVES KLETT I., DRECHSLER K.: Designing technical tessellations. pp. 305–322. 3