Computational Design of Transforming Pop-up Books

NAN XIAO and ZHE ZHU, TNList, Department of Computer Science and Technology, Tsinghua University, China
RALPH R. MARTIN, School of Computer Science and Informatics, Cardiff University, UK
KUN XU*, JIA-MING LU, and SHI-MIN HU, TNList, Department of Computer Science and Technology, Tsinghua University, China

We present the first computational tool to help ordinary users create *transforming pop-up* books. In each transforming pop-up, when the user pulls a tab, an initial flat 2D pattern, i.e. a 2D shape with a superimposed picture, such as an airplane, turns into a new 2D pattern, such as a robot. Given the two 2D patterns, our approach automatically computes a 3D pop-up mechanism that transforms one pattern into the other; it also outputs a design blueprint, allowing the user to easily make the final model. We also present a theoretical analysis of basic transformation mechanisms; combining these basic mechanisms allows more flexibility of final designs. Using our approach, inexperienced users can create models in a short time; previously, even experienced artists often took weeks to manually create them. We demonstrate our method on a variety of real world examples.

CCS Concepts: • **Computing methodologies** → *Mesh geometry models*;

Additional Key Words and Phrases: paper art, pop-up, transforming pop-up, mechanism design

ACM Reference Format:
Nan Xiao, Zhe Zhu, Ralph R. Martin, Kun Xu, Jia-Ming Lu, and Shi-Min Hu. 2017. Computational Design of Transforming Pop-up Books. *ACM Trans. Graph.* 1, 1, Article 1 (January 2017), 15 pages. https://doi.org/10.1145/XXXXXXX.YYYYYYY

1 INTRODUCTION

Most youngsters of today are familiar with Transformers toys produced by Takara and Hasbro, in which a toy can change shape, for example from a plane into a robot, using a combination of mechanisms. Youngsters of an earlier generation were perhaps more familiar with pop-up books—when their pages were opened, objects such as castles or animals would pop up from the page. More recently, these two ideas were brought together in such publications as Reinhart's best-selling book [2013], which contains sophisticated examples of transforming pop-ups. When the user pulls a tab, an initial flat 2D pattern, i.e. a 2D shape with a superimposed picture, turns into a new 2D pattern, standing up from the page. We illustrate one of Reinhart's transforming pop-ups in Figure 2.

Creating such a book is a big challenge even for experienced paper artist-engineers. Designing the necessary mechanisms to create the desired effects, without interference, can take even a talented professional many weeks of trial and error. Typically, the designer first creates an initial pop-up from a combination of mechanisms to transform between the two desired patterns, and then adjusts the patterns to fit the pop-up. An iterative process of adjusting the pop-up and the 2D patterns then follows until satisfactory, aesthetically pleasing results are achieved.

Researchers have already considered automating the design process for pop-up books [Le et al. 2014; Li et al. 2011, 2010]. However, the kind of pop-ups so far considered depict only one meaningful object: an arbitrary flat design turns into the desired object. In contrast, transforming pop-ups depict two different meaningful objects, linked by a transformation. Due to this intrinsic difference, existing automated pop-up design approaches cannot be directly applied to the design of transforming pop-ups.

This paper considers automating the design of transforming pop-ups. We first study the principles of assembling a transforming pop-up. We discuss several basic mechanisms and types of linkages for combining them. We then show how a transforming pop-up can be readily composited by combining appropriate mechanisms via these linkages.

Our main contribution, apart from the discussion of suitable mechanisms and linkages, is a fully automatic algorithm for creating a transforming pop-up that approximately transforms one user-provided pattern into another. An optimization-based approach attempts to match the user given patterns in the initial and final states as closely as possible, while at the same time ensuring as a hard constraint that interference between parts of the pop-up cannot occur. After optimization, the pop-up is painted with colored textures. Finally, a fabrication template is generated, allowing the user to physically realize the pop-up.

2 RELATED WORK

2.1 Computational pop-ups

There are several kinds of pop-up art, such as *V-style* pop-ups [Li et al. 2011], *multi-style* pop-ups [Ruiz et al. 2014], *sliceform* pop-ups [Le-Nguyen et al. 2013] and *transforming* pop-ups [Glassner 2002b]. They can be categorized as one-state pop-ups and two-state pop-ups [Glassner 2002b]. [Le-Nguyen et al. 2013; Li et al. 2011; Mitani and Suzuki 2004;

*corresponding author

Authors' addresses: Nan Xiao, cloudia@126.com; Zhe Zhu, ajex1988@gmail.com, TNList, Department of Computer Science and Technology, Tsinghua University, China; Ralph R. Martin, MartinRR@cardiff.ac.uk, School of Computer Science and Informatics, Cardiff University, UK; Kun Xu, xukun@tsinghua.edu.cn; Jia-Ming Lu, loyaveforever@gmail.com; Shi-Min Hu, shimin@tsinghua.edu.cn, TNList, Department of Computer Science and Technology, Tsinghua University, China.

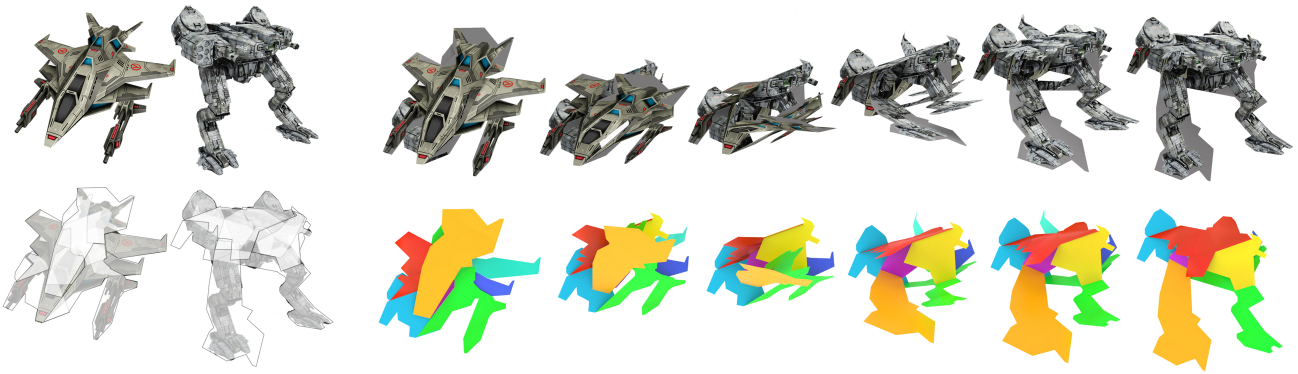


Fig. 1. A 3D transformation between a fighter jet and a battle robot. Top left: two input patterns. Top right: a transformation sequence with textured patches. Bottom left: layer relationships between patches in untransformed and transformed states. Bottom right: the transformation sequence showing each patch in a different color. Photos © TurboSquid.

(a) (b)

Fig. 2. Transforming pop-up by Matthew Reinhart, showing (a) car form (before transformation), and (b) robot form of the Hornet Transformer (after transformation). Original artworks © Matthew Reinhart.

Mitani et al. 2003; Ruiz et al. 2014] consider one-state pop-ups: their transformation process turns a 2D folded shape with an invisible pattern into a 3D structure. With regard to two-state pop-ups, while [Glassner 2002b] mentioned transforming pop-ups with a pull-tab, he did not give an automatic method for their construction. In computer graphics, the study of pop-ups has evolved from semi-automatic design [Okamura and Igarashi 2009] to fully-automatic construction. To assist in the design process, Lee et al. [1996] simulate the motion of the pop-up pieces when the pop-up book is opened; in particular, angles between intersecting pieces are calculated. However, this approach provides an analysis tool rather than an automatic design tool. To help automate the design process, Glassner [2002a] considered the motion paths of the vertices of slit and V-fold mechanisms in pop-ups. Various further assistance tools, such as collision detectors and printing generators, were also provided in [Glassner 2002b]. Rather than providing an analysis tool or assistance tools, Li et al. [2010] proposed an automated approach to generate pop-ups from input 3D models. In [Li et al. 2010], two sets of parallel paper pieces are used to construct the final structure. Later, Li extended his work to V-style pop-ups [2011] for which the final structure is composed of V-fold structures. However, in [Li et al. 2010], a voxel discretization step is used, leading to discretization artifacts along the boundaries of the final pop-ups; these can be visually displeasing. To improve the quality of the output paper designs, Lee et al. [2014] proposed an approach that better preserves shape details by taking contours, surfaces, and volumes of the input models into consideration. Other types of pop-ups have also been investigated, including sliceform pop-ups [Le-Nguyen et al. 2013; Mitani and Suzuki 2003], in which the structure is made of two sets of parallel paper patches slotted together, and multi-style pop-ups [Ruiz et al. 2014], which combine several previously studied pop-up elements: step-fold, tent-fold, V-fold and box-fold. However, *transforming pop-ups* have yet to be considered from a computational point of view. Since previous works have almost exclusively focused on *one-state* pop-ups, their approaches are in general unsuited to transforming pop-ups.

2.2 Mechanical design & analysis

Spurred by the recent interest in 3D printing, researchers in computer graphics have turned their attention to mechanism design and analysis. Given a designer specified geometry and motion for a toy, Zhu et al. [2012] provide an approach to automatically generate a mechanism assembly located in a box below a toy character. Kinematic simulation [Winder et al. 2009] ensures that the fabricated toy functions properly. However, their approach can only handle simple motions such as linear, ellipsoidal and circular motions. Coros et al. [2013] allow more complex, user defined motions, but they must still be cyclic. In [Coros et al. 2013; Zhu et al. 2012] the user defines motion curves, each of which indicates the motion direction of a specific part of the mechanism.

An alternative high-level approach to interaction is to let the user provide functional relations [Koo et al. 2014]. Given an approximate 3D model, the user annotates functional relationships such as *part A supports parts B and C*. Based on such information, their approach optimizes the geometry of the model to produce a working design. In our work, we use

a few basic mechanisms which can be regarded as generalized forms of 4-revolute-joint linkages ($4R$ linkages) [Norton 2011]. In mechanism design theory, a $4R$ linkage is the simplest movable closed chain linkage comprising four bodies connected in a loop by four joints. It has one degree of freedom, which means it can be driven by one body. This is appropriate to our problem, allowing the pop-up to be controlled by a single tab pulled by the user. A $3R$ linkage is self-locked (i.e. has no degrees of freedom), while an nR linkage for $n > 4$ has $n - 3$ degrees of freedom, giving too much freedom. Since we need the paper pieces to have complex shapes to model the input shapes, our problem differs from traditional mechanical design and assembly problems, and cannot readily be solved using existing methods of mechanism design [Joskowicz and Sacks 1994].

2.3 Folding & combination of objects

Folding, and combination by attachment, are two ways to generate new objects from old. In our problem, the union of several paper patches forms the input in its laid-flat form (i.e. on the surface of the book). A compound mechanism is formed by combining several basic mechanisms. Folding is inherently suitable for paper, as it is a thin sheet material; folding has also been used in e.g. furniture design [Li et al. 2015], the design of working mechanical objects [Koo et al. 2014], as well as the design of iris folding patterns [Igarashi et al. 2016].

To generate 3D models composed of interlocking planar pieces [McCrae et al. 2011; Schwartzburg and Pauly 2013], rules were devised for combining planar pieces under geometric constraints imposed by fabrication and assembly. Since cubes and boxes can be stacked readily for transportation and storage, Zhou et al. [2014] considered how to transform a 3D object into a cube, by defining several types of joints between voxels and searching a voxel-tree to find a folding sequence. Inspired by their work, we define our own types of joints as well as a tree structure of joints. We obtain our final result by searching a tree to optimize the mechanism. Another popular method is to define a connection graph [Xin et al. 2011], but this approach is unsuited to our problem: it focuses on 3D models.

3 TRANSFORMING POP-UPS

The problem to be solved is as follows. A pattern is a 2D shape with a texture (i.e. coloring). Given two patterns as input, we wish to compute a mechanism that can smoothly transform in 3D between one pattern and the other (see the top row of Figure 1). We also wish to automatically generate a blueprint, allowing a user to fabricate the transforming pop-up by gluing and assembling printed pieces of paper. The pipeline of our approach is provided in Figure 10, but before explaining it, we first consider some theoretical background ideas.

The key components of a transforming pop-up are *patches*, *hinges*, and *slits*:

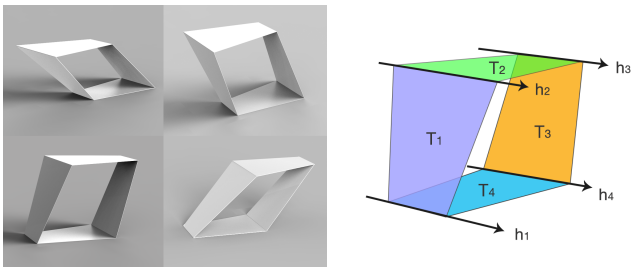


Fig. 3. Parallel mechanism: (a) four intermediate states of a parallel mechanism during transformation, (b) patches, hinges and their relationships.

A **patch** is a planar polygon, embodied physically as a piece of paper, whose two sides can be differently textured (i.e. coloured with a drawing). A number of patches are used in combination to depict a pattern.

A **hinge** is a shared axis along which two patches are joined, and about which they can rotate relative to one another.

A **slit** is a straight line cut in the interior of a patch, of infinitesimal width. Another patch can pass through such a slit.

A *linkage* is the coupling between two elements of a mechanism that transfers the movement between them.

Following [Li et al. 2011], we regard paper as a rigid material that can rotate around hinges, and assume for simplicity that it has zero thickness and weight.

From a mechanical view, a transforming pop-up is composed of several *basic mechanisms* and other *functional elements*, which together we call *basic elements*. These basic elements are linked together using a few predefined types of linkage. When users pull (or push) a tab, the pop-up turns from an initial flat 2D pattern to the new 2D pattern via a 3D transformation. The transformation is driven by the tab, and is transferred to the whole pop-up through the linkages. We refer to the initial state and the final state of the pop-up as the *untransformed state* and the *transformed state*, respectively.

We next consider the basic mechanisms used in Section 3.1, and functional elements in Section 3.2, as well as the linkages in Section 3.3. We then introduce geometric parameters, and the parameter space of the transforming pop-up, used for reasoning about the mechanisms, in Section 3.4. We discuss their use in the design optimization algorithm in Section 4.

3.1 Basic Mechanisms

We use three basic mechanisms based on the $4R$ linkage, a fundamental closed loop mechanism [Norton 2011] with 4 hinges and one degree of freedom. We now describe each in turn.

Parallel mechanism. A parallel mechanism (PM for short) is composed of four quadrilateral patches T_i ($1 \leq i \leq 4$) connected cyclically by four hinges as shown in Figure 3. In a PM, all four hinges are parallel, while patches T_1 and T_3, and T_2 and T_4, are parallel in pairs.

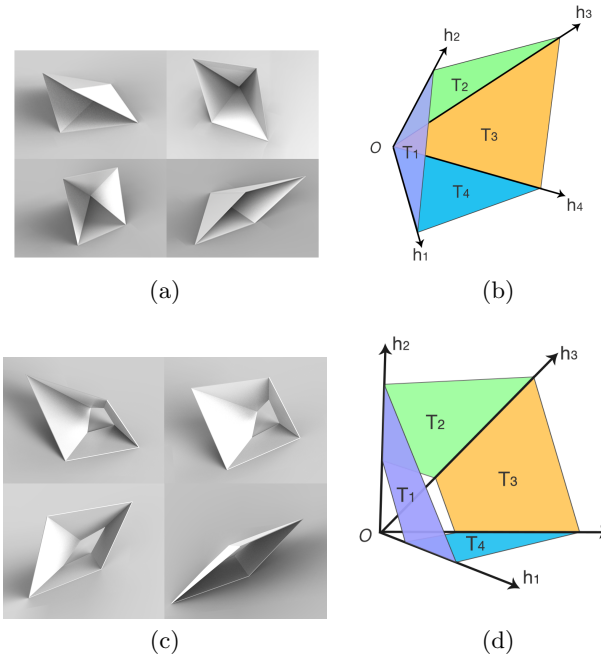


Fig. 4. V-fold mechanism: (a) four intermediate states of a typical V-fold mechanism during transformation, (b) patches, hinges and their relationships, (c), (d) another V-fold mechanism in which the patches do not include the pyramidal vertex.

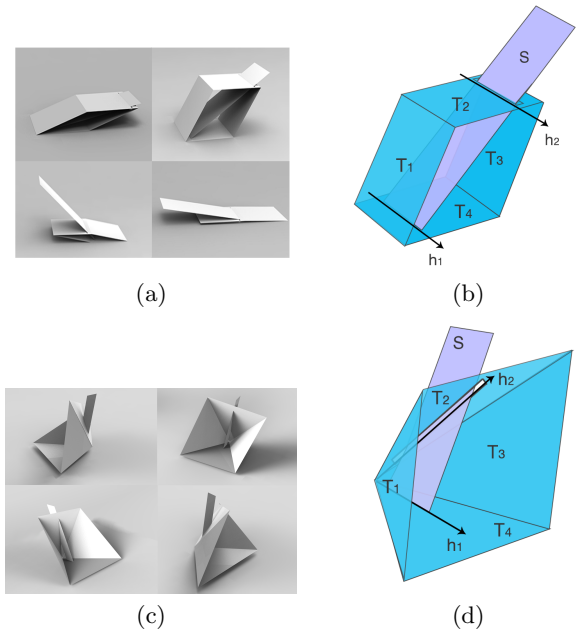


Fig. 5. Slide mechanism

V-fold mechanism. As shown in Figures 4(a) and 4(b), a V-fold mechanism (VM for short) is also composed of four patches T_i ($1 \leq i \leq 4$) with four hinges h_i ($1 \leq i \leq 4$). In this case, the (extended) lines containing the four hinges intersect at a single point, the *pyramidal vertex* of the VM. Denoting the angle between hinges h_i and h_j by α_{ij}, it is necessary that $\alpha_{14} = \alpha_{23}$ and $\alpha_{12} = \alpha_{34}$. Note that the pyramidal vertex need not actually belong to any of the patches—see Figures 4(c) and 4(d). In the latter case, the patches are quadrilaterals rather than triangles.

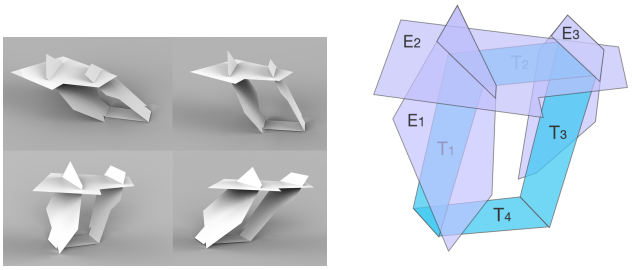
Slide mechanism. See Figure 5. A slide mechanism (SM for short) is composed of one rectangular patch, the *slide patch*, a hinge, and a slit; it is associated with a PM or a VM. The slit is placed in one of the patches of the associated mechanism, and the hinge is placed on another patch of the same associated mechanism. The slide patch passes through the slit. An SM must satisfy several constraints. Firstly, the width of the slit should exceed the maximum width of that part of the slide patch that passes through the slit during transformation. Secondly, the length of the slide patch should be longer than the maximum distance between the slit and the hinge during transformation. Thirdly, when associated with a PM, both the slit and the hinge of the SM must be parallel to the hinge of the PM; when associated with a VM, both the line of the slit and the line of the hinge must intersect the pyramidal vertex of the VM. The first constraint ensures that the slide mechanism will not jam, the second ensures that the slide patch cannot fall out of the slit, and the final constraint ensures that the hinge and slit are coplanar. As the linked mechanism moves, the degree to which the slide patch protrudes through the slit changes. Slide mechanisms allow the slide patch to change from exposed to hidden, or vice versa.

3.2 Functional Elements

In addition to the above basic mechanisms, we also make use of other functional elements, which we now describe.

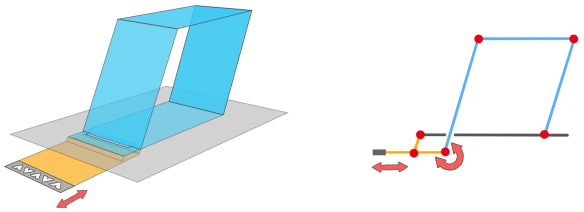
Extended patch. As defined, patches of a PM or a VM are restricted to quadrilaterals or triangles, which would limit the range of shapes the pop-up could represent. To allow more flexibility, we allow further polygons lying in the same plane to be attached to the patches of a PM or VM. An augmented patch of this kind is referred to as an *extended patch* (EP for short). In our implementation, we restrict extended patches to having six or fewer vertices, to constrain the number of parameters during pop-up meachnism optimization. An example is illustrated in Figure 6.

Ground patch and Pull-tab. The transforming pop-up needs to have one ground patch and one pull-tab to function properly. The ground patch remains static during transformation; it also serves as the reference frame. We select a *root mechanism* from amongst the PMs and VMs, then the ground patch is selected as one of the patches belonging to the root mechanism. The ground patch can be extended for better aesthetics, and generally forms a large part of the page of the book. The pull-tab is a rectangular patch that the user pulls (or pushes) to actuate transformation of the pop-up. It



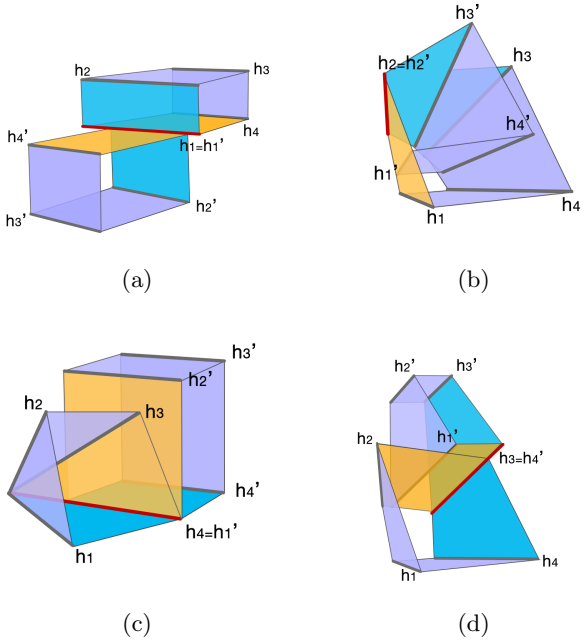
(a) (b)

Fig. 6. Three extended patches attached to a parallel mechanism.



(a) (b)

Fig. 7. (a) A pull-tab. (b) Side view of this pull-tab.



(a) (b) (c) (d)

Fig. 8. Hinge linkages. (a): Opposite hinge linkage. (b): Inner hinge linkage. (c), (d): Supplementary hinge linkages.

is associated with a small parallel mechanism which is linked to the root mechanism through an opposite hinge linkage: see Figure 7. In any mechanism, *pulling* the tab makes the transformation go in one direction, and *pushing* the tab makes it go in the opposite direction. The designer decides for any given mechanism whether the initial tab direction is pull or push, when manually adding the tab.

3.3 Linkage Tree

The overall transforming pop-up is constructed by linking basic elements (i.e. basic mechanisms and functional elements). Transformation is actuated by the pull-tab, whose movement is transferred from one element to another by linkages which make the whole pop-up move. Note that linkages are directional, as one element is active and drives the other which is passive. We now consider the allowed types of linkage.

Slide linkage. Linkages may go from a PM or a VM to an SM. The movement is transmitted via the shared hinge and slit.

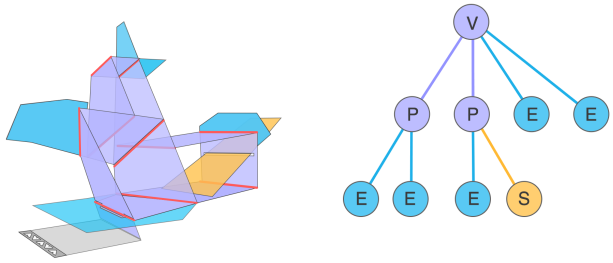
Extended patch linkage. Linkages may also go from a PM or a VM to an EP. The movement is transmitted through the shared plane.

Hinge linkage. A hinge linkage transmits movement from one PM or VM to another PM or VM. Two such linked basic mechanisms share a hinge line, and a pair of neighboring patches is restricted to be coplanar. Note that the two basic mechanisms do not necessarily have common end points on the hinge. According to the relative positions of the two basic mechanisms, hinge linkages can be subdivided into three kinds as shown in Figure 8. When one basic mechanism is placed opposite the other (see Figure 8(a)), the result is an *opposite hinge linkage*. If one basic mechanism is placed inside the other (see Figure 8(b)), an *inner hinge linkage* results. Finally, if one basic mechanism is placed adjacent to the other (see Figures 8(c) and 8(d)), a *supplementary hinge linkage* results.

Linkage Tree. The compound mechanism of a transforming pop-up forms a *linkage tree*, in which each node represents a basic mechanism or a functional element, and each edge represents a linkage. The basic mechanism connected to the pull-tab is the root node. EPs and SMs can only serve as leaf nodes, while PMs and VMs can be leaf or internal nodes. The pull-tab and the ground patch are not represented in the linkage tree. An example linkage tree is illustrated in Figure 9.

3.4 Parameter Space

The above ideas describe the possible components and their connectivity which go to make a pop-up. However, to fully determine a transforming pop-up, we also need to find values for various geometric parameters, which determine the positions of patch vertices, hinges and slits, etc. Although these positions change when transforming, it suffices to describe them in the untransformed state to fully determine the pop-up. The parameters are adjusted to achieve a working mechanism meeting the designer's goals as closely as possible by means of the optimization algorithm given in Section 4. However, we first consider how these parameters are defined in the ground plane.



(a) (b)

Fig. 9. Linkage tree. (a) A simple compound mechanism containing one V-fold mechanism and two parallel mechanisms (purple), several extended patches (blue) and a slide patch (yellow). (b) Corresponding linkage tree. At the root is a V-fold mechanism linked to two parallel mechanisms and two extended patches. One of the parallel mechanisms is linked to two extended patches; the other is linked to an extended patch and a slide patch.

A PM, as shown in Figure 3, has 4 hinges and 8 vertices. To define the 4 parallel hinge lines in the 2D ground plane, we need 2 parameters to define the first line, and 2 more parameters to define the second and the third lines (i.e. distance to the first line); the fourth line is determined by the three other lines. Since the 8 vertices are located on the 4 hinge lines, 8 additional parameters are needed to define the 8 vertices (i.e. one for each vertex). In total, the parameter space of a PM is thus 12D (i.e. 12-dimensional).

Similarly, a VM, as shown in Figure 4, has one pyramidal vertex, 4 hinges and 8 vertices. Two parameters suffice to define the pyramidal vertex. Since the 4 hinge lines intersect at the pyramidal vertex, 3 parameters are needed for the first, second, and third lines (i.e. an angle with respect to the pyramidal vertex for each); the 4 fourth line is determined by the 3 other lines. Since the 8 vertices are located on the 4 hinge lines, 8 further parameters are needed to define the 8 vertices. In total, the parameter space of a VM is thus 13D.

An SM, as shown in Figure 5, has a hinge, a slit and a slide patch. An SM is associated with a PM *or* a VM, and is constrained so that the hinge line and slit line are parallel to the hinge of the associated PM, *or* the hinge line and slit line intersect at the pyramidal vertex of the associated VM. These constraints mean 2 parameters are needed to define the hinge line and the slit line (for a PM, we store the distance to one hinge of the PM, while for a VM, we store the angle with respect to the pyramidal vertex of the VM). We also need 2 parameters to define the 2 end vertices of the hinge. Since the slide patch is a rectangle, 1 parameter is needed to define its length. Note that we do not need any parameters to define the two end vertices of the slit since they are implicitly determined by the movement of the slide patch during transformation. Thus, in total, the parameter space of an SM is 5D.

Finally, the parameter space for an extended patch is up to 12D, since it has at most 6 vertices and each vertex lies in 2D.

The overall set of geometric parameters of the whole pop-up is obtained by concatenating the geometric parameters of each of its basic mechanisms and extended patches. However, care must be taken with hinge linkages. Since two linked mechanisms share a hinge line, we do not need to store the hinge line in both mechanisms. For all 3 types of hinge linkage, adding a linkage between two basic mechanisms reduces the dimension of the parameter space by two. As an example, consider the pop-up in Figure 9. It contains 2 PMs, 1 VM, 1 SM, 5 EPs, and two hinge linkages. Hence, the dimension of its parameter space is $12 \times 2 + 13 + 5 + 12 \times 5 - 2 \times 2 = 98$.

In summary: a transforming pop-up is fully determined by its linkage tree, its components and their connectivity, and the geometric parameters described above.

We next consider, given two 2D patterns, how to determine such a transforming pop-up, by optimization of both its linkage tree and geometric parameters.

4 ALGORITHM

4.1 Overview

A transforming pop-up moves from the untransformed state to the transformed state; each is a 2D pattern. Our goal is to generate a pop-up, that appears similar to the user given *source pattern* in its untransformed state, and to the user given *target pattern* in its transformed state. During transformation, we must ensure, as a hard constraint, that the mechanisms remain collision-free. The generated transforming pop-up should also satisfy two soft constraints: a shape constraint and a flip constraint. The first is that the *shapes* of the pop-up in its untransformed and transformed states should match the source and target patterns as closely as possible (we consider the appearances of these shapes later). Secondly, any patches, or parts thereof, which are visible (i.e. not occluded by other patches) in both untransformed and transformed states, should flip over during transformation. If their front appears in the untransformed state, their back should appear in the transformed state. This allows the visible parts of the patches in untransformed and transformed states to be painted differently with the textures of source and target patterns, respectively.

Our overall algorithm has four steps: an optimization step, a patch refinement step, a texture painting step, and a blueprint generation step.

The optimization step produces an initial pop-up. An energy function models both the shape constraint and the flip constraint, as well as ensuring freedom from collisions as a hard constraint. To limit the number of linkage tree configurations considered to reasonable bounds, we restrict the maximal number of basic mechanisms and extended patches that can be used. We obtain the pop-up with minimal energy by considering all possible configurations of the linkage

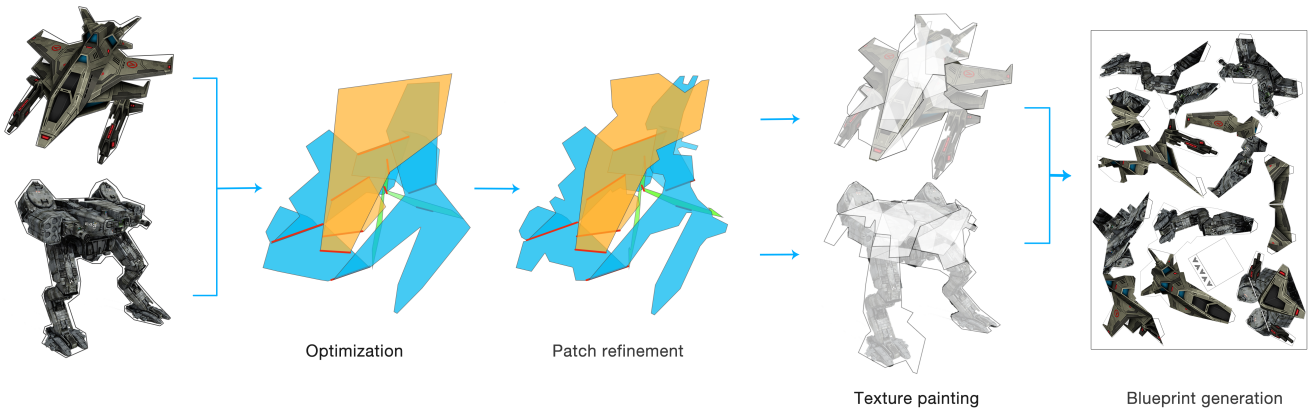


Fig. 10. Pipeline of our approach.

tree, and for each configuration, optimizing over the geometric parameters. Since the energy function is non-linear and the parameter space is high-dimensional, we use a genetic algorithm.

As the patches in the transforming pop-up are polygons with only a few vertices, this optimization step restricts the pop-up to representing simple, approximate shapes. To determine a more elaborate shape for the pattern, we use a refinement step. In this step, some patches are cropped to better match the shapes of the user given patterns. Since we only crop patches and do not add any new material to the pop-up, the mechanism structures are preserved, as are the constraints—as long as we do not crop too much: see Section 4.3.

After cropping, we paint textures onto the patches of the pop-up in both untransformed and transformed states, as needed by the source and target patterns.

Finally, we output a blueprint for users to construct the transforming pop-up. We print all parts of the pop-up onto paper with suitable attachment tabs for assembly. Users are able to fabricate the pop-up by cutting the parts out, and gluing them together.

We now consider these steps in further detail.

4.2 Optimization

4.2.1 Energy function. We define the optimization problem in terms of minimising an energy function $E(x)$, where x represents the linkage configuration and geometric parameters of the transforming pop-up. For simplicity of presentation, we leave the dependence on x implicit in the following.

Ensuring freedom from collisions is enforced as a hard constraint. Given a configuration x, we first check for collision during transformation, and if detected, we give the energy function E a value certain to be greater than the energy for any non-colliding configuration. Our implementation uses the OBB-tree algorithm [Gottschalk et al. 1996] for collision detection, like other recent mechanical modeling works such as [Zhu et al. 2012]. To avoid collision with the ground, the designer may optionally include a dummy ground patch of large enough area to enclose the design during collision checking. If the goal is to place the pop-up inside a book, this option should be enabled, but for a free-standing pop-up, it is unnecessary.

If there is no collision, the energy function E is defined as the sum of a shape term and a flip term:

$$E = \lambda_s E_s + (1 - \lambda_s) E_f. \tag{1}$$

The shape term E_s represents the shape constraint, i.e. it favors shapes of pop-up for which the untransformed and transformed states are good matches to the user given patterns. The flip term E_f represents the flip constraint, penalizing any patch areas which do not flip and are visible in both untransformed and transformed states. λ_s controls the relative contributions of the two terms, and is set to 0.4.

Shape term. The shape term accounts for how well the pop-up in the untransformed and transformed states S_1 and S_2 match the shapes of the user given source and target patterns U_1 and U_2. The shape matching cost includes both a contour matching cost and an area matching cost:

$$E_s = E_c + E_a. \tag{2}$$

The former is computed using the *shape context* method [Belongie et al. 2002], and is defined as:

$$E_c = D(S_1, U_1) + D(S_2, U_2), \tag{3}$$

where $D(\cdot)$ denotes the shape context matching cost. In our implementation, we sample 100 points on each contour to perform this computation.

The area matching cost measures the difference of region coverage between the pop-up and user given patterns. It is defined to be:

$$E_a = \sum_{i=1,2} \lambda_a \|S_i \setminus U_i\| + (1 - \lambda_a) \|U_i \setminus S_i\|, \tag{4}$$

where $\setminus$ is the set difference operator, $\|\cdot\|$ denotes the fractional area of a region, i.e. its area divided by the total area of all patches, and λ_a is a weight. $S_i \setminus U_i$ are *redundant regions*, i.e. regions covered by the pop-up but not covered by the user given pattern. $U_i \setminus S_i$ are *unrepresented regions*, i.e. regions covered by the user given pattern but not covered by the pop-up. Redundant regions are relatively unimportant, since the patch refinement step performed later can crop them. However, unrepresented regions will show the ground patch as they will not be part of the transforming pop-up. Thus, we set $\lambda_a = 0.1$ to more strongly penalize unrepresented regions. We will evaluate the effect of these two terms in Section 5.2.

Flip term. Since visible regions in untransformed and transformed states are painted with source and target pattern textures respectively, to avoid conflicts in texture painting, we should avoid regions which show the same side of the paper in both untransformed and transformed states. The flip term is simply defined as the sum of the fractional areas of all regions which violate this constraint:

$$E_f = \sum_i \|R_i\|, \tag{5}$$

where R_i is any region of a patch which is visible from the same side in both states.

4.2.2 Energy minimization. We first describe our basic approach to solving the minimization problem, then discuss how to improve its performance.

Basic approach. To minimize the energy function in Equation 1 we iterate over all possible linkage tree configurations, up to a certain maximum size. To limit the number of configurations to reasonable bounds, we restrict the number of basic mechanisms and extended patches that can be used. We observe that transforming pop-ups made by artists [Reinhart 2013] generally contain 3 or 4 basic mechanisms, so we restrict the number of basic mechanisms to 5 or fewer. The number of extended patches is restricted to 12 or fewer—more would lead to pop-ups with smaller pieces, which are harder to construct and more fragile, as well as leading to a more time-consuming optimization problem.

Simulation of the mechanism during optimization is driven by the pull-tab. The patch of the mechanism connected to the pull-tab makes an angle with the pull-tab which is varied in small steps from 0° to 180° to operate the mechanism. We can compute the angles at other hinges using simple geometric considerations. Since the energy function is highly non-linear, we use a genetic algorithm (GA) to optimize the geometric parameters. In order to constrain the search space, all the geometric parameters, including positions of vertices and hinges, are encoded using integer based discrete pixel locations, with a resolution of 1024×1024 pixels. In initialization, all parameters are randomly generated. The fitness function for a configuration x is given by $-E(x)$. Mutation is performed by adding a normally distributed value to each parameter in the individual. Single point crossover and tournament selection are used. In our implementation, we run the GA 10 times, each time for 50 generations, using a population of 1000 individuals, with a crossover probability of 0.4 and mutation probability of 0.3.

Acceleration. Due to the high dimensionality of the parameter space, directly using a GA as described above is time consuming, taking tens of hours to generate reasonable output. Our initial experiments showed that after a few generations, the GA obtains an adequate global structure for the whole mechanism, but it takes much longer to optimize precise vertex positions of each patch. Using this observation, we use a greedy strategy [Won and Lee 2016] to accelerate the search. Specifically, during optimization we detect *good patch vertices* as the GA iterates, and fix them to reduce the dimensionality of the solution space. Patch refinement (as explained in the next section) is done before detection, to increase the potential number of good patch vertices. *Good patch vertices* are ones satisfying any of the criteria below:

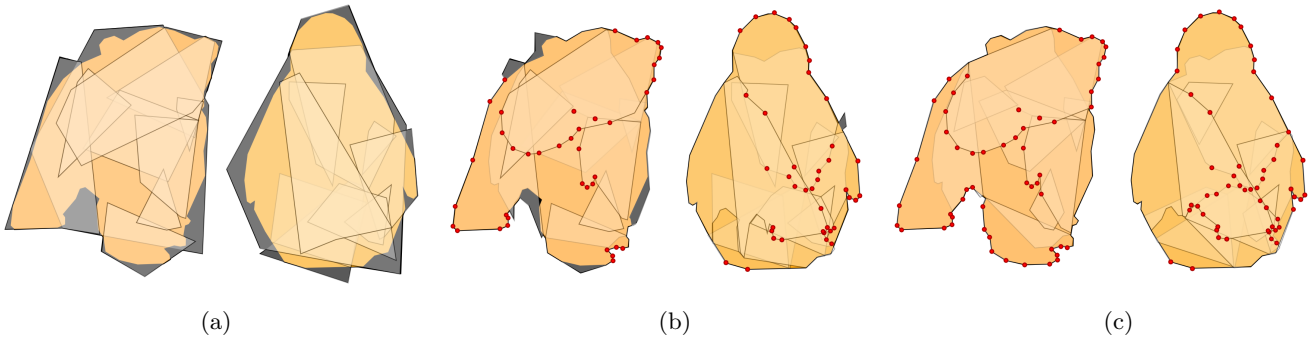
- Vertices belonging to any patch which flips, and which lie on the boundaries of the two input patterns.
- Vertices belonging to any patch which flips, and which lie on the boundary of the input pattern in one state, and inside the target shape in the other state.
- Vertices belonging to any patch which does not flip, and which lie on the boundary of the input pattern in one state, and are occluded by other patches in the other state.

We illustrate greedy acceleration and good patch vertices in Figure 11. Further speed up is achieved by implementing the above algorithm using CUDA on a GPU. As it is highly parallel, the GPU implementation runs much faster. Typically, the greedy strategy provides a 5–10× speed up, while GPU implementation provides another 10× speed up compared to a CPU-based implementation. Overall, this acceleration reduces the processing time from tens of hours to a few minutes.

4.3 Patch refinement

The initial transforming pop-up obtained by the basic optimization approach has patches which are polygons with only a few vertices, so the pop-up can only represent coarse, simple shapes. Hence, it is inevitable that there are still some redundant regions (covered by the pop-up but not by the user given patterns) and unrepresented regions (covered by the user given patterns but not by the pop-up). The purpose of this step is to reduce the size of the *redundant* regions by refining some of the patches. Since we use a high weight to penalize *unrepresented* regions, they rarely appear in our initial result.

We deal with redundant regions for the untransformed state and for the transformed state in turn. In principle, for the untransformed state, we crop all redundant regions, except for those parts which belong to visible regions in the transformed state, since cropping those regions would affect the appearance of the transformed state. Redundant regions in the transformed state are cropped in an analogous way. In theory, the above described cropping scheme might potentially break the structure of the mechanism: a hinge



(a) (b) (c)

Fig. 11. Greedy acceleration. (a) is an intermediate result from the brute force GA. Patch refinement gives the result in (b). Afterwards, some boundary vertices are fixed(marked in red). Using our greedy acceleration strategy gives the final result in (c).

might be completed cropped, one patch might be cropped into two disconnected parts, or the slider in a slide mechanism might be cropped so as to be too short. However, we have not observed the above issues arising in practice, so we do not incorporate additional schemes to avoid them; manual adjustment could potentially be needed if those issues happen in unforeseen examples.

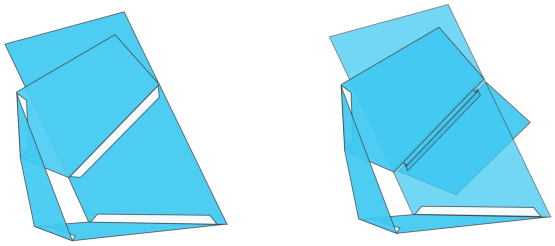
4.4 Texture painting

After determining the mechanism and set of patches, we must paint the user provided textures onto the patches. As well as those patches visible in the untransformed and transformed states, all other patches should also be painted, in order to make the intermediate appearance look more natural during the transformation process. Both faces of each patch must be painted. The texture mapping process is simple, as we only need to directly map the textures of source and target patterns onto the patches without any deformation. For each patch, we must decide which texture to use, i.e. the source pattern or the target pattern.

This is done for each face in turn. We sequentially decide:

- If the face is visible (or partially visible) in both states: the parts only visible in the untransformed or transformed state are painted using source or target pattern respectively. Any conflicting parts (i.e. visible in both states) and parts invisible in both states are painted using the source pattern.
- If the face is only visible (or partially visible) in one state (untransformed or transformed): the whole face is painted using a single texture (source or target pattern respectively).
- Otherwise, the face is invisible in both states: the whole face is painted using the source pattern.

Care must be taken for redundant regions and unrepresented regions. We fill any redundant regions with the average color of source and target patterns, to ensure consistency with the texture inside the pattern. Unrepresented regions become a part of the ground patch and are not included in the transformation of the pop-up, but are still painted using mapped textures from source or target patterns.



(a) (b)

Fig. 12. Simple glued hinge, and opposite hinge.

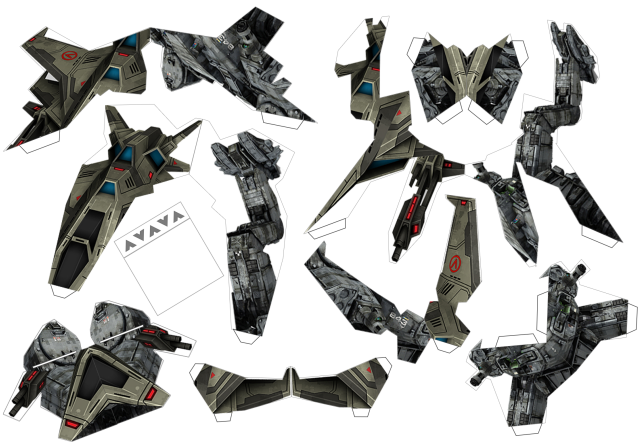


Fig. 13. A blueprint.

4.5 Blueprint generation

We finally generate a blueprint to help the user fabricate the transforming pop-up. Remember that a patch and any further polygons attached to it form an extended patch. See

Table 1. Numbers and types of mechanisms, and extended patches, used in each case in Fig. 15. PM: parallel mechanisms, VM: V-fold mechanisms, SM: slide mechanisms, EP: extended patches.

Fig.	Num. of Mech.	PM	VM	SM	EP
15 (a)	1	0	1	0	3
15 (b)	2	1	1	0	5
15 (c)	3	1	2	0	6
15 (d)	4	1	3	0	7
15 (e)	5	2	3	0	9

Figure 8(a): the two blue patches, and the two yellow patches, each form an extended patch. Both front and reverse of a given extended patch are printed on the same side of paper, touching each other. The user folds the cut-out shape along the line where front and reverse meet, and glues it to itself to give the two sided appearance. (We do not use duplex printing since registration is typically imperfect).

When printing the patches, we also print tabs to allow the patches to be joined. *Inner* and *supplementary* hinges can be made simply by gluing a tab on one patch to an adjacent patch. *Opposite* hinges (where planes cross) require a slit for manufacturing; they are made as shown in Figure 12. The pull-tab is manually designed and is also printed as part of the blueprint. A typical blueprint is illustrated in Figure 13.

5 EVALUATION AND RESULTS

Our approach has been implemented in C++ on a PC with an Intel Xeon E5-2620 2.0GHz CPU with 32GB memory, and in CUDA on an NVIDIA Titan X GPU with 16GB memory. We have validated our approach on a variety of real-world cases.

5.1 Evaluation: number of basic mechanisms

Our choice of using up to 5 basic mechanisms provides a good balance between quality and speed, as well as suitability of design for manufacture and robustness in use. In Figure 15, we give results when using different numbers of basic mechanisms, for a fixed pair of input patterns, while Table 1 gives the numbers and kinds of basic mechanisms used in each case.

The result in Figure 15(a) uses only one mechanism resulting in obvious and obtrusive redundant regions. The results in Figures 15(b) and 15(c) are much better, but still suffer from inaccurate boundaries. The results in Figure 15(d) and Figure 15(e) are both visually pleasing, but Figure 15(e) is more accurate than Figure 15(d). Using more than 5 mechanisms, the computation time increases dramatically, and fabrication also becomes harder.

5.2 Evaluation: energy terms

The shape context matching cost E_c and area matching cost E_a both play important roles in energy minimization, as we now demonstrate by considering some intermediate states during optimization of the caterpillar-to-butterfly example.

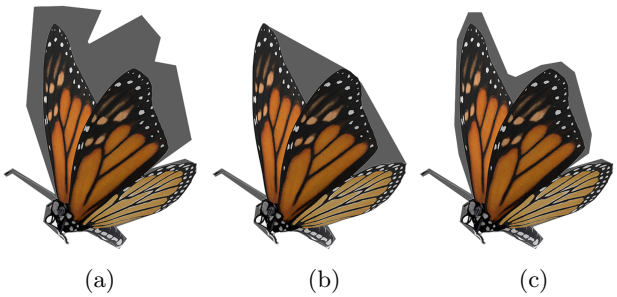


Fig. 14. Intermediate results with different context matching and area matching costs (see Table 2).(photo © PixelSquid)

Table 2. Shape context matching cost and area matching cost for several examples.

Fig.	Shape context cost	Area cost
14 (a)	0.1069	0.3424
14 (b)	0.1273	0.1507
14 (c)	0.0992	0.1575

Table 3. For each example, we give: the example name, the corresponding figure number in the paper, and the sizes of the redundant regions (relative to the size of the original shape) in untransformed and transformed states, respectively.

Example Name	Fig.	redundant region size
Car–robot	19	0.027/0.022
Egg–chick	19	0.005/0.07
Sphinx–mask 1	21	0.008/0.012
Heart–rose	16	0.016/0.822

Figure 14 shows various intermediate states, while the corresponding energies are given in Table 2. Comparing Figures 14(a) and 14(b), in the latter, the area cost is much lower, so much of the redundant region disappears, but on the other hand, the overall shape is not much like the target shape. On the other hand, in Figure 14(c), the shape cost is lower, resulting in a shape which more closely follows that of the desired result. Using both terms helps to ensure that the final design has only small redundant regions, but when they are unavoidable, the resulting shapes look similar to those intended by the designer, making the redundancy less noticeable.

In detail, we give the sizes of redundant regions for some examples in Table 3. For the extreme example transforming a heart into a rose (Figure 16), the relative size of redundant regions is much larger than in other successful examples. Size of redundant regions provides a good indication of the quality of the output transforming pop-up.

5.3 Results

Figure 17 shows three simple pop-ups generated by our system, transforming a disk into a triangle, a rectangle into a disk,

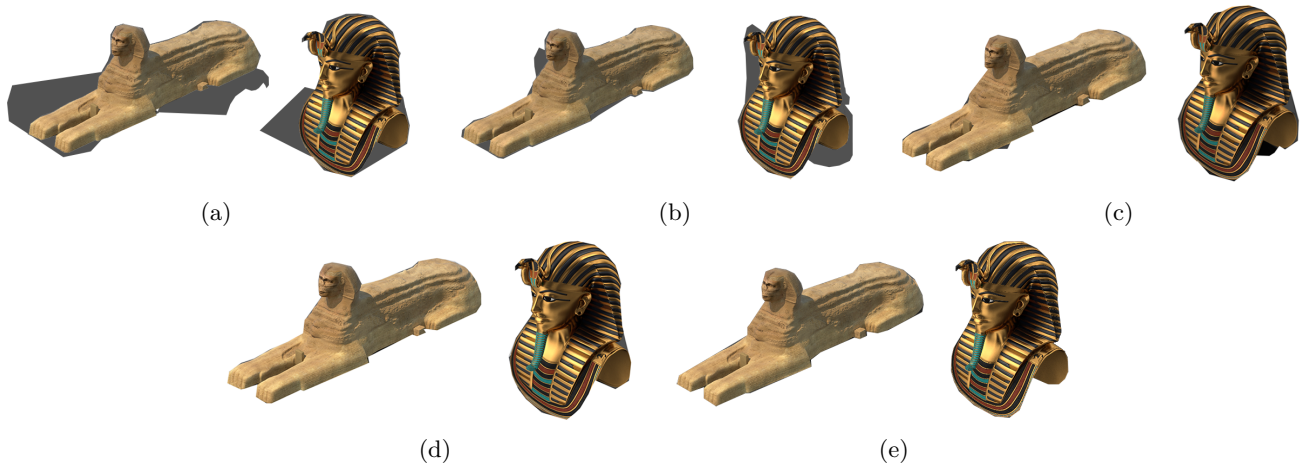


Fig. 15. Results using varying numbers of mechanisms: from (a) to (e), 1 to 5 mechanisms.(photos © PixelSquid)

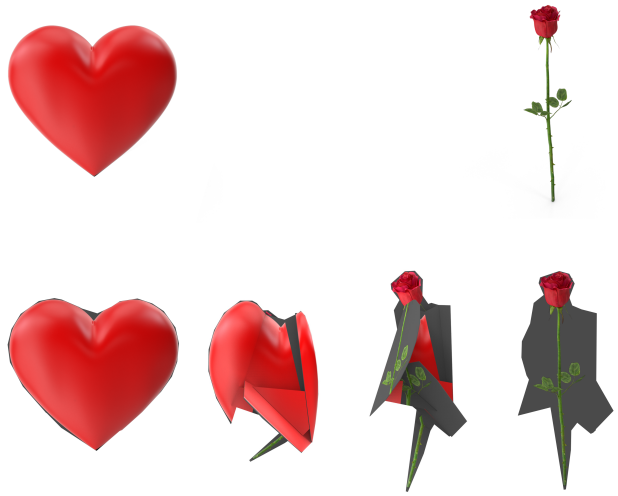


Fig. 16. Extreme example, with a very thin target pattern. Above: input patterns. Below: computed pop-up. Photos © PixelSquid.

and a rectangle into a triangle, respectively. The in-between shapes change in a reasonable way in these cases.

Figure 18 shows two additional pop-ups generated by our system, transforming between randomly chosen shapes from the MPEG-7 shape matching dataset. The top one transforms frog-19 into bell-3, and the bottom one transforms tree-14 into flatfish-1. Our system demonstrate well on such randomly chosen shapes.

Figure 19 shows a further complex example, transforming an egg into a chick. The redundant regions in the egg to chick case are mainly by the chick's feet, which are thin and highly concave.

Figure 20 shows two examples with and without ground collision checking, transforming a caterpillar into a butterfly. Both results give interesting transformations.

Figure 21 shows two more examples, transforming a hot air balloon into a space shuttle, and the Great Sphinx of Giza into King Tutankhamun's Burial Mask. For each example, we show two different transforming pop-ups. Since a GA is a random algorithm, different runs typically provide different solutions, allowing the user to generate alternative possibilities if the first attempt is not considered sufficiently aesthetically pleasing.

Finally, we also show an extreme case in Figure 16, which transforms a heart into a (very thin) rose. Since the two shapes differ greatly, our approach inevitably produces large redundant regions.

We give statistics for each example in Table 4, including the number of basic mechanisms used, and the computation times required by the original GA algorithm and the GPU based greedy GA algorithm. An animation of the transformation for each example can be found in the supplementary video.

6 CONCLUSION AND DISCUSSION

This paper has presented an approach to automatically generate transforming pop-ups. Using three basic mechanisms with simple structure, and modifying and combining them in different ways, allows us to use optimization to devise structures which smoothly transform between two 2D patterns. The final output is a blueprint for constructing the mechanism, so that the user can easily make a physical working pop-up. We have demonstrated the effectiveness of our approach with many shape pairs. Results show that our approach can generate visually pleasing transforming pop-ups.

Limitations and future work. Since we use polygonal patches in our approach, we cannot readily work with shapes with curved boundaries. From the fabrication point of view, we have not considered how to efficiently use the area of the paper—we do not guarantee our generated blueprint to be the most economic solution, nor do we optimize the tabs for strength. Another limitation is that we assume paper thickness

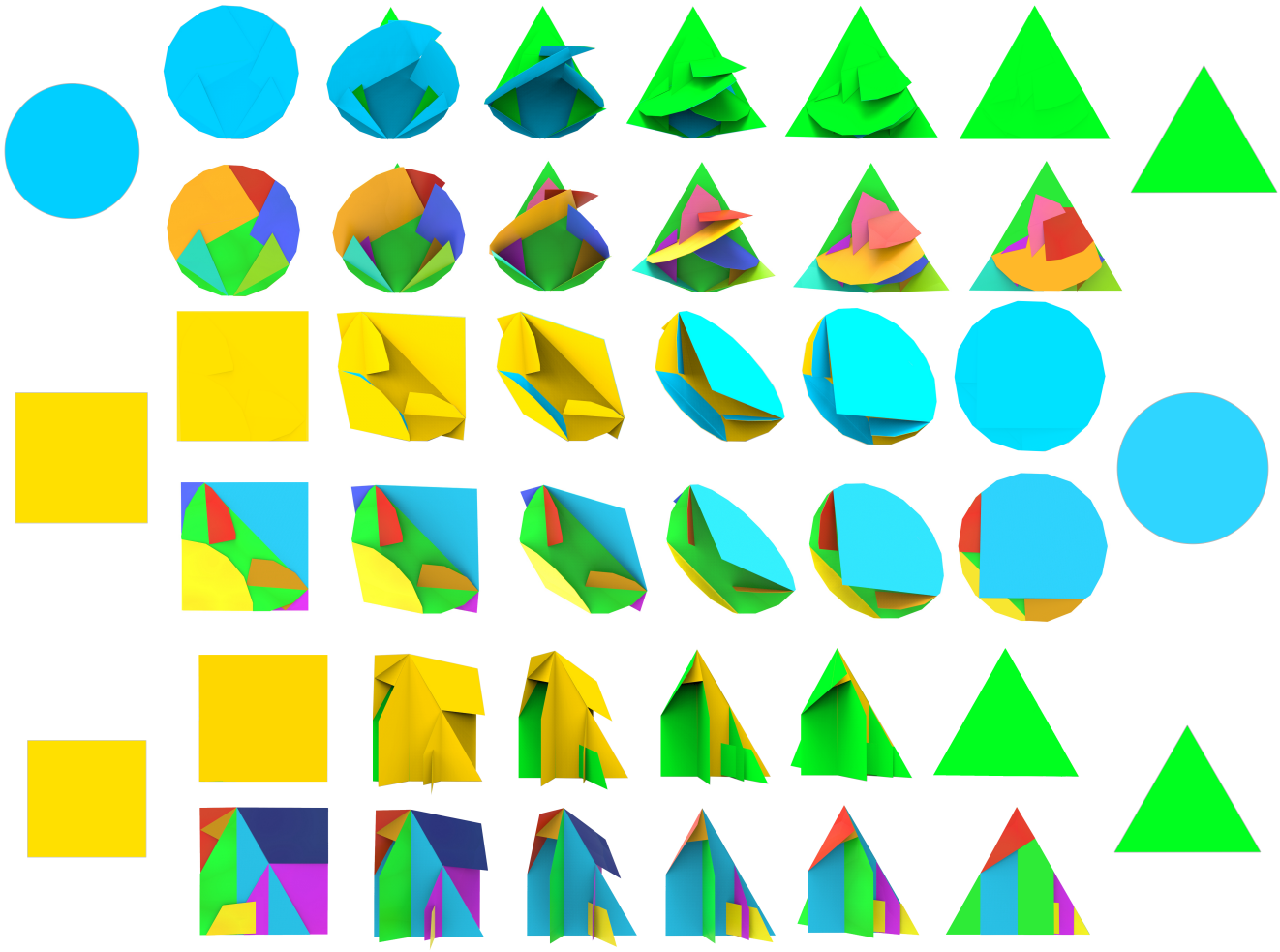


Fig. 17. Three simple examples. Top to bottom: transforming a disk into a triangle, a rectangle into a disk and a rectangle into a triangle. Top: the appearance seen by the user. Below: individual patches colored to allow visualization of the mechanism.

to be zero, which might cause the fabricated mechanism to fold improperly if there are too many layers near hinges. A possible way to address this could be to add a thickness term to the energy function. Finally, our current algorithm does not consider pattern semantics. To do so, a possible approach might be to ask the designer to provide a meaningful segmentation of each shape, indicating correspondences, and then to optimize matching of segments instead of matching of shapes.

In future, we also plan to investigate transformations between 2D patterns and 3D shapes, as well as between 3D shapes.

ACKNOWLEDGMENTS

We thank Haozhi Huang and Bin Liu for their help in producing the supplementary video. This work was supported by the National Key Technology R&D Program (Project Number 2017YFB1002604), the Natural Science Foundation of China(Project Number 61521002), Research Grant of Beijing Higher Institution Engineering Research Center, Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and an EPSRC Travel Grant.

REFERENCES

S. Belongie, J. Malik, and J. Puzicha. 2002. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 4 (April 2002), 509–522.

Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. 2013. Computational Design of Mechanical Characters. *ACM Trans. Graph.* 32, 4, Article 83 (July 2013), 12 pages.

Andrew Glassner. 2002a. Interactive pop-up card design, part 1. *IEEE Comput. Graph. Appl* 20, 1 (2002), 79–86.

Andrew Glassner. 2002b. Interactive pop-up card design, part 2. *IEEE Comput. Graph. Appl* 20, 2 (2002), 74–85.

S. Gottschalk, M. C. Lin, and D. Manocha. 1996. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *Proceedings of SIGGRAPH 96 (Annual Conference Series)*. 171–180. https://doi.org/10.1145/237170.237244

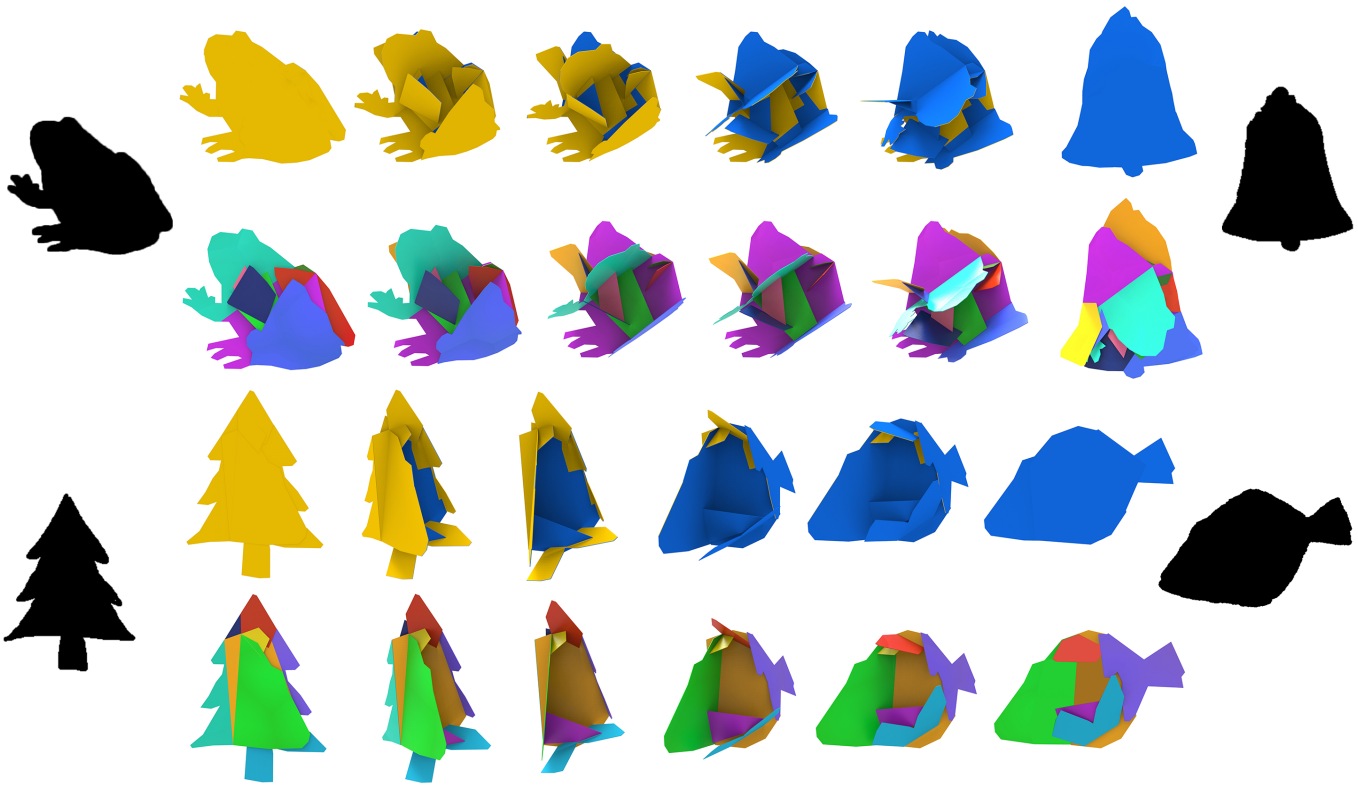


Fig. 18. Two examples transforming between randomly chosen shapes from the MPEG-7 dataset. Top to bottom: transforming frog-19 into bell-3, and tree-14 into flatfish-1.

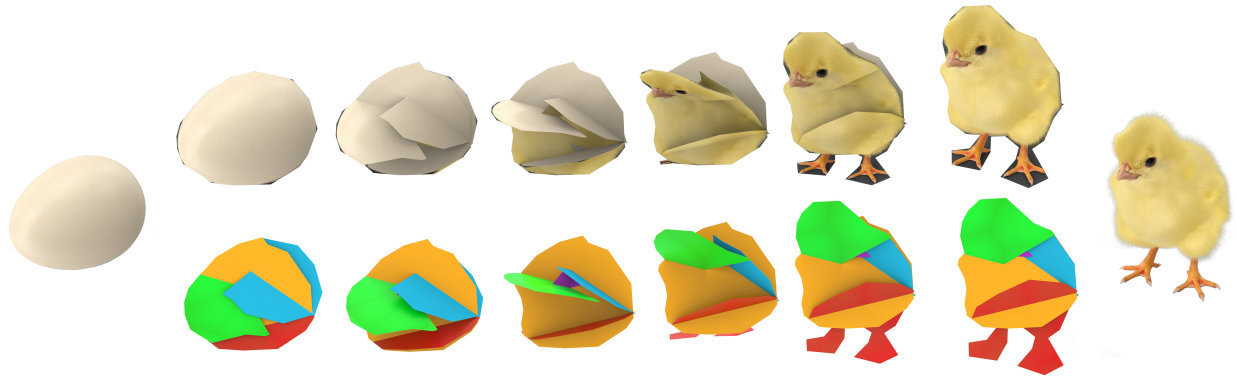


Fig. 19. A further example, which transforms an egg into a chick (photos © PixelSquid).

Yuki Igarashi, Takeo Igarashi, and Jun Mitani. 2016. Computational design of iris folding patterns. *Computational Visual Media* 2, 4 (01 Dec 2016), 321–327. https://doi.org/10.1007/s41095-016-0062-4

L. Joskowicz and E. Sacks. 1994. Configuration space computation for mechanism design. In *In Proceedings of IEEE International Conference on Robotics and Automation*. 1080–1087. https://doi.org/10.1109/ROBOT.1994.351215

Bongjin Koo, Wilmot Li, JiaXian Yao, Maneesh Agrawala, and Niloy J. Mitra. 2014. Creating Works-like Prototypes of Mechanical Objects. *ACM Trans. Graph.* 33, 6, Article 217 (Nov. 2014), 9 pages.

Sang N. Le, Su-Jun Leow, Tuong-Vu Le-Nguyen, Conrado Ruiz, and Kok-Lim Low. 2014. Surface and Contour-Preserving Origamic Architecture Paper Pop-Ups. *IEEE Trans. Vis. Comput. Graph.* 20, 2 (Feb. 2014), 276–288.

Tuong-Vu Le-Nguyen, Kok-Lim Low, Conrado R. Ruiz Jr., and Sang N. Le. 2013. Automatic Paper Sliceform Design from 3D Solid Models. *IEEE Trans. Vis. Comput. Graph.* 19, 11 (2013), 1795–1807.

Y.T. Lee, S.B. Tor, and E.L. Soo. 1996. Mathematical modelling and simulation of pop-up books. *Computers & Graphics* 20, 1 (1996), 21 – 31.

Honghua Li, Ruizhen Hu, Ibraheem Alhashim, and Hao Zhang. 2015. Foldabilizing Furniture. *ACM Trans. Graph.* 34, 4, Article 90 (July 2015), 12 pages.

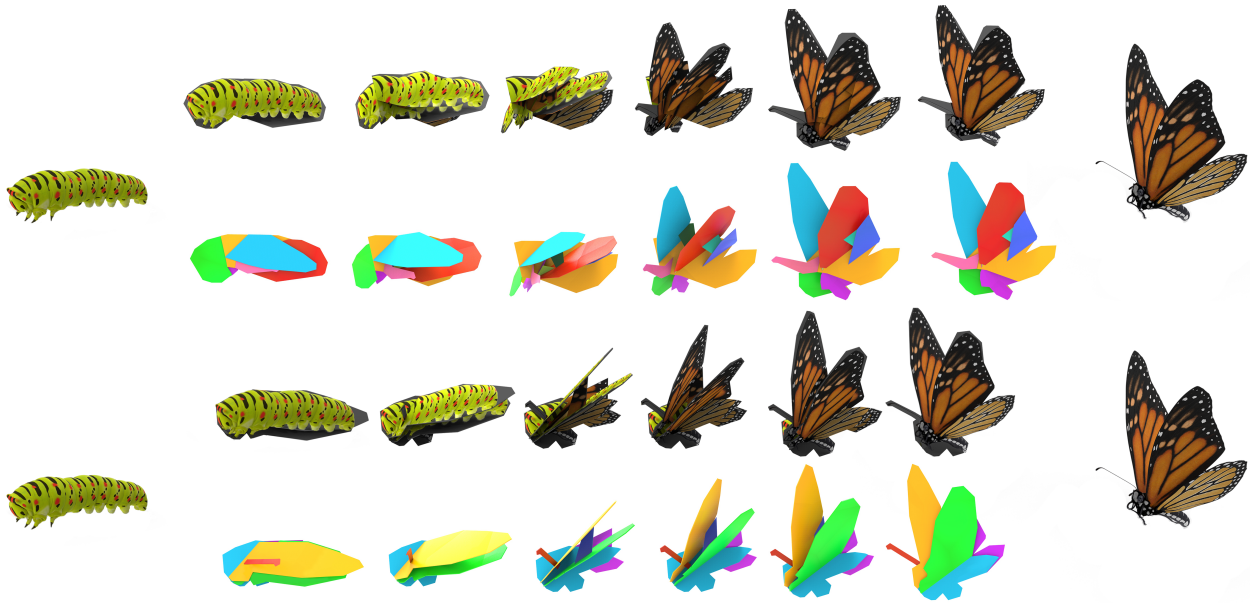


Fig. 20. Examples with (above) and without (below) ground collision checking, showing transformation of a caterpillar into a butterfly (photos © PixelSquid).

Table 4. Each example lists: its name, the corresponding figure number in the paper, the numbers of each kind of basic mechanism, and extended patches, used—PM: parallel mechanism, VM: V-fold mechanism, SM: slide mechanism, EP: extended patch, whether the basic algorithm (N) or the accelerated algorithm (Y) is used, and computation time.

Example	Fig.	PM	VM	SM	EP	Ac.	Tm.
Fighter–robot	1	1	2	0	12	N	16h
Disk–triangle	17	1	3	0	8	Y	9min
Square–disk	17	1	2	0	5	Y	9min
Square–triangle	17	1	2	1	5	Y	5min
frog–bell	18	1	4	0	8	Y	18min
tree–flatfish	18	0	4	0	5	Y	6min
Egg–chick	19	0	2	0	5	Y	8min
Cater.–butterf. 1	20	3	2	0	7	Y	17min
Cater.–butterf. 2	20	0	2	1	5	N	14h
Ball.–shuttle 1	21	1	3	0	9	N	14h
Ball.–shuttle 2	21	1	3	0	8	Y	10min
Sphinx–mask 1	21	1	3	0	7	N	16h
Sphinx–mask 2	21	1	2	0	6	Y	10min

Xian-Ying Li, Tao Ju, Yan Gu, and Shi-Min Hu. 2011. A Geometric Study of V-style Pop-ups: Theories and Algorithms. *ACM Trans. Graph.* 30, 4 (2011), 98:1–10.

Xian-Ying Li, Chao-Hui Shen, Shi-Sheng Huang, Tao Ju, and Shi-Min Hu. 2010. Popup: automatic paper architectures from 3D models. *ACM Trans. Graph.* 29, 4 (2010), 111:1–9.

James McCrae, Karan Singh, and Niloy J. Mitra. 2011. Slices: A Shape-proxy Based on Planar Sections. *ACM Trans. Graph.* 30, 6, Article 168 (Dec. 2011), 12 pages. https://doi.org/10.1145/2070781.2024202

Jun Mitani and Hiromasa Suzuki. 2003. Computer Aided Design for 180-degree Flat Fold Origamic Architecture with Lattice-type Cross Sections. *Journal of Graphic Science of Japan* 37, 3 (2003), 3–8. https://doi.org/10.5989/jsgs.37.3_3

J. Mitani and H. Suzuki. 2004. Computer aided design for Origamic Architecture models with polygonal representation. In *Proceedings of the Computer Graphics International*. 93–99.

J. Mitani, H. Suzuki, and H. Uno. 2003. Computer aided design for origamic architecture models with voxel data structure. *Transactions of Information Processing Society of Japan* 44, 5 (2003), 1372–1379.

Robert L. Norton. 2011. *Design of Machinery*. McGraw-Hill Education; 5 edition (March 30, 2011).

Sosuke Okamura and Takeo Igarashi. 2009. An Interface for Assisting the Design and Production of Pop-Up Card. *Lecture Notes in Computer Science* 5531, 2 (2009), 68–78.

Matthew Reinhart. 2013. *Transformers: The Ultimate Pop-up Universe*. LB Kids; Pop edition.

Conrado R. Ruiz, Sang N. Le, Jinze Yu, and Kok-Lim Low. 2014. Multi-style paper pop-up designs from 3D models. *Computer Graphics Forum* 33, 2 (2014), 487–496. https://doi.org/10.1111/cgf.12320

Yuliy Schwartzburg and Mark Pauly. 2013. Fabrication-aware Design with Intersecting Planar Pieces. *Comput. Graph. Forum* 32, 2 (2013), 317–326.

Brian G Winder, Spencer P Magleby, and Larry L Howell. 2009. Kinematic representations of pop-up paper mechanisms. *Journal of Mechanisms and Robotics* 1, 2 (2009).

Jungdam Won and Jehee Lee. 2016. Shadow Theatre: Discovering Human Motion from a Sequence of Silhouettes. *ACM Trans. Graph.* 35, 4, Article 147 (July 2016), 12 pages. https://doi.org/10.1145/2897824.2925869

Shiqing Xin, Chi-Fu Lai, Chi-Wing Fu, Tien-Tsin Wong, Ying He, and Daniel Cohen-Or. 2011. Making Burr Puzzles from 3D Models. *ACM Trans. Graph.* 30, 4, Article 97 (July 2011), 8 pages. https://doi.org/10.1145/2010324.1964992

Yahan Zhou, Shinjiro Sueda, Wojciech Matusik, and Ariel Shamir. 2014. Boxelization: Folding 3D Objects into Boxes. *ACM Trans. Graph.* 33, 4, Article 71 (July 2014), 8 pages. https://doi.org/10.1145/2601097.2601173

Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. 2012. Motion-guided Mechanical Toy Modeling. *ACM Trans. Graph.* 31, 6, Article 127 (Nov. 2012), 10 pages.

Received December 2016; revised July 2017; final version October 2017; accepted October 2017

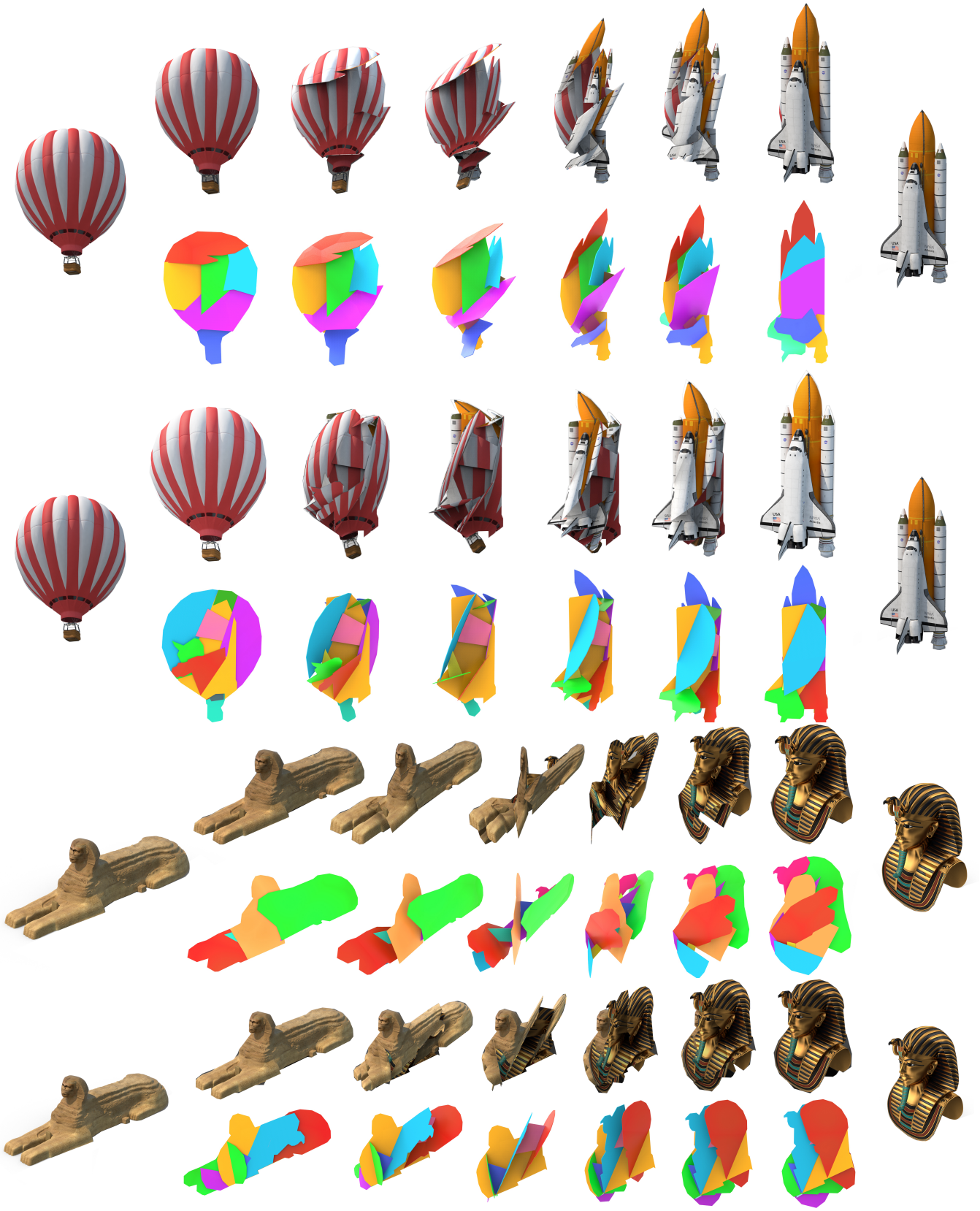


Fig. 21. Alternative solutions, showing two different transformations from a balloon into a space shuttle, and from the Great Sphinx of Giza into King Tutankhamun's Burial Mask. Photos © PixelSquid.