Computational Design of Transforming Pop-up Books

NAN XIAO and ZHE ZHU
TNList, Tsinghua University
and
RALPH R. MARTIN
Cardiff University
and
KUN XU and JIA-MING LU and SHI-MIN HU
TNList, Tsinghua University

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, standing up from the page. 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.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric algorithms, languages, and systems*

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

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, via 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 1.

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 [Li et al. 2010; Li et al. 2011; Le et al. 2014]. 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, and a transformation between them. Due to this intrinsic difference, existing automated pop-up design approaches cannot be directly applied to the design of transforming pop-ups.

(a) Car form, before transformation (b) Robot form, after transformation

Fig. 1. Transforming pop-up by Matthew Reinhart, showing car and robot forms of the Hornet Transformer.

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 easily 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 well 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 color textures. Finally, a construction template is generated, allowing the user to fabricate 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 classified into two categories: one-state pop-ups, and two-state pop-ups [Glassner 2002b]. [Li et al. 2011; Ruiz et al. 2014; Le-Nguyen et al. 2013; Mitani and Suzuki 2004; Mitani et al. 2003] consider one-state pop-ups: their transformation

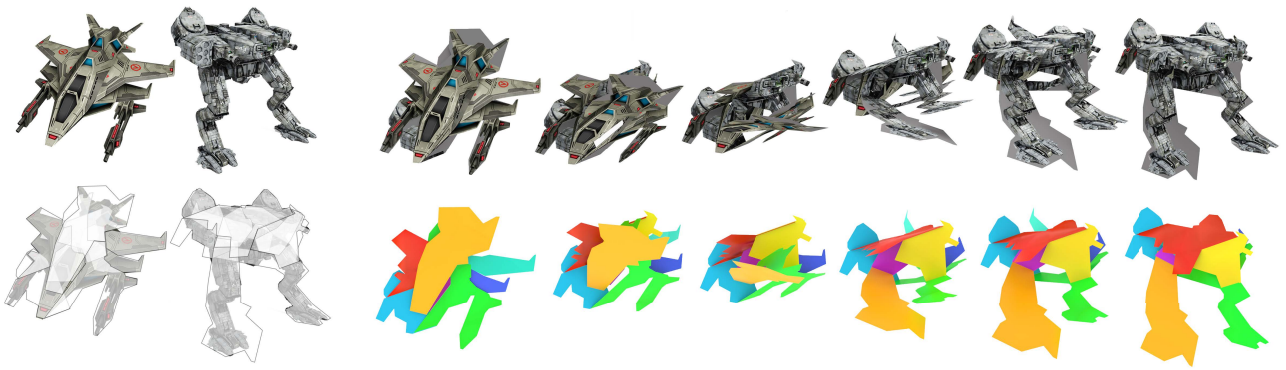


Fig. 2. A 3D transformation between a fighter jet and a battle robot. Top left: two input patterns. Top right: a transformation sequence with textured patches. Bottom left: layer relationships between patches in untransformed and transformed states. Bottom right: the transformation sequence showing each patch in a different color.

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 solution 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, their approach is really 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] where 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 in the boundaries of the final pop-ups, which can be visually displeasing. To improve the quality of the output paper work, 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. To assist in toy design, given a user specified geometry and motion of the toy, Zhu et al. [2012] provide an approach to automatically generate a mechanism assembly located in a box below a character. Kinematic simulation [G. Winder and L. Howell 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, although they must still be cyclic. In [Zhu et al. 2012; Coros et al. 2013] 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 this information, their approach optimizes the geometry of the model to produce a working design. In our work, we use some 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, as the pop-up is 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, and so too much freedom. Since we need the paper pieces to have complex shapes to model the input shapes, our problem is different 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 when lying flat (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] as well as the design of working mechanical objects [Koo et al. 2014].

To generate 3D models composed of interlocking planar pieces [Schwartzburg and Pauly 2013; McCrae et al. 2011], 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 case as it focuses on 3D models.

3. TRANSFORMING POP-UP

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 perform 3D transformation between one pattern and the other. We also wish to automatically generate a blueprint, so that a user can fabricate the transforming pop-up by gluing and assembling printed pieces of paper.

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

A **patch** is a planar polygon. Physically, it is a piece of paper, whose two sides can be differently textured (i.e. coloured with a drawing). A number of patches are used together 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 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 a tab, the pop-up turns from an initial flat 2D pattern to new erect 2D pattern through 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.

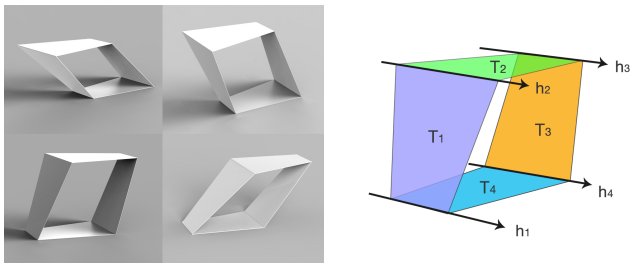
We now turn to consider the basic mechanisms used in Section 3.1, and functional elements in Section 3.2, as well as the linkages in Section 3.3. We then introduce geometric parameters, and the parameter space of the transforming pop-up, used for reasoning about the mechanisms in Section 3.4. We discuss 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.

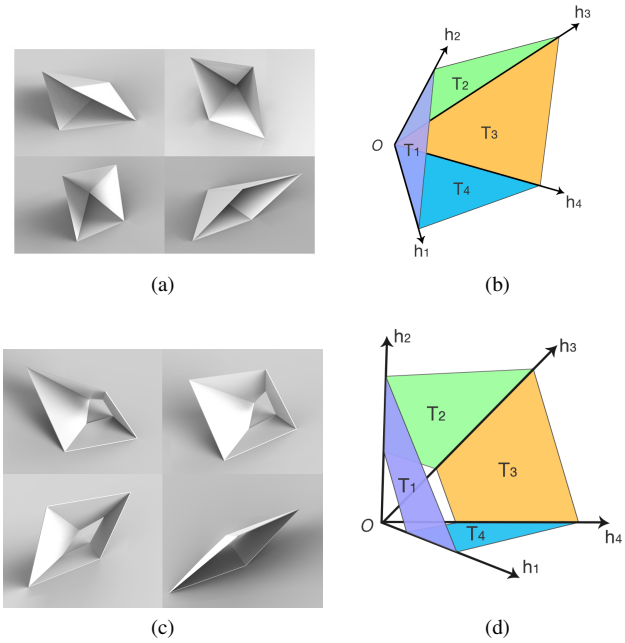
Parallel mechanism. A parallel mechanism (PM for short) is composed of four quadrilateral patches T_i, $1 \leq i \leq 4$ connected cyclically by four hinges as shown in Figure 3. In a PM, the four hinges are parallel to one another, while patches T_1 and T_3, and T_2 and T_4, are parallel in pairs.

V-fold mechanism. As shown in Figures 4(a) and 4(b), a V-fold mechanism (VM for short) is composed of four patches T_i4, $1 \leq i \leq 4$ with four hinges h_i, $1 \leq i \leq 4$. In this case, the extended lines of the four hinges intersect at one point, which is referred to as the *pyramidal vertex* of the VM. Furthermore, 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 be on any of the patches—see Figures 4(c) and 4(d). In the latter case, the patches are quadrilaterals rather than triangles.



(a) (b)

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



(a) (b)

(c) (d)

Fig. 4. V-fold mechanism: (a) four stages 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.

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 be larger than the maximum width of the part of the slide patch that passes through the slit during transformation. Secondly, the length of the slide patch should be longer than the 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 get jammed, while the second ensures that the slide patch

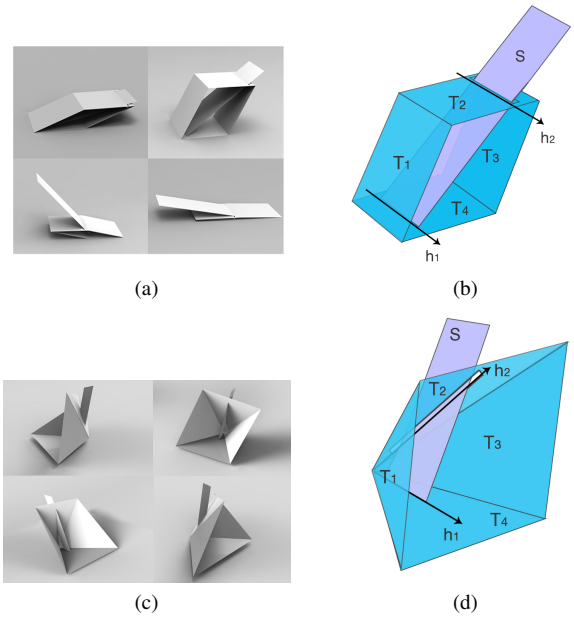


Fig. 5. Slide mechanism

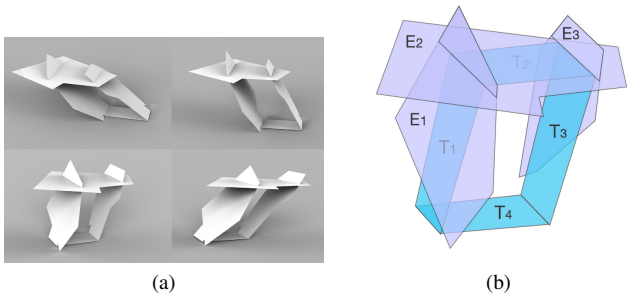


Fig. 6. Three extended patched attached to a parallel mechanism.

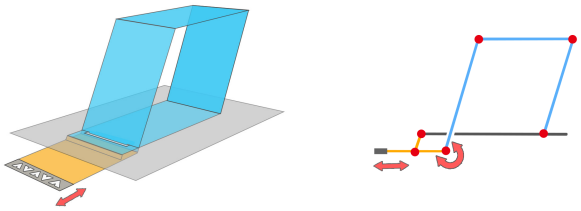


Fig. 7. (a) Illustration of a pull-tab. (b) Side view of a pull-tab.

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 several 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 limits the range of shapes the pop-up can 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 those with six or fewer vertices. 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, and 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 is associated with a small parallel mechanism which is linked to the root mechanism through an opposite hinge linkage: see Figure 7. This actually shows a *push*-tab—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 user decides for any given mechanism whether the initial tab direction is pull or push, when manually adding the tab.

3.3 Linkage Tree

The whole transforming pop-up is constructed by linking mechanical elements and functional elements. Transformation is actuated by the pull-tab, whose movement is transferred from one element to another via linkages to 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 mechanisms do not necessarily have common end points on the hinge. According to the relative positions of the two mechanisms, hinge linkages can be subdivided into three kinds as shown in Figure 8. When one mechanism is placed opposite the other (see Figure 8(a)), we have an *opposite hinge linkage*. When one mechanism is placed inside the other (see Figure 8(b)), we have an *inner hinge linkage*. Finally, when one mechanism is placed adjacent to the other (see Figures 8(c) and 8(d)), we have a *supplemental hinge linkage*.

Linkage Tree. The compound mechanism of a transforming pop-up forms a *linkage tree*, where 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. EP and SM mechanisms can only serve as terminal nodes, while PM and VM can be either terminal or non-terminal nodes. The pull-tab and the ground patch are not represented in the linkage tree. An example linkage tree is illustrated in Figure 9.

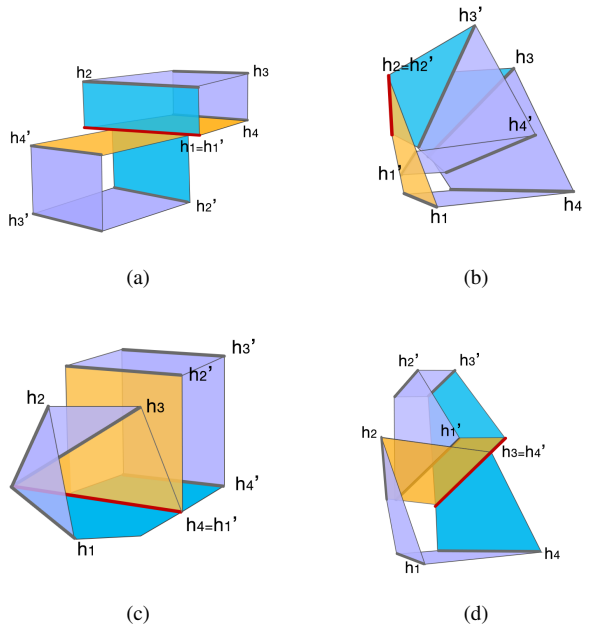


Fig. 8. Hinge Linkages. (a): Opposite hinge linkage. (b): Inner hinge linkage. (c), (d): Supplementary hinge linkages.

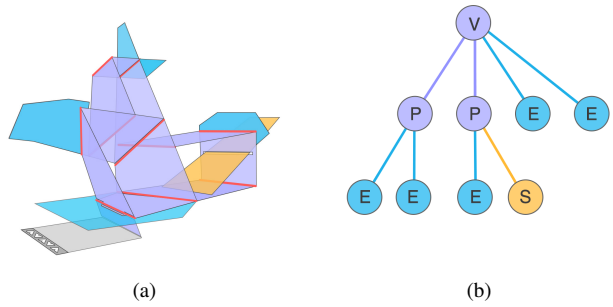


Fig. 9. Linkage tree. (a) A simple compound mechanism containing one V-fold mechanism and two parallel mechanisms (purple), some extended patches (blue) and a slide patch (yellow). (b) Corresponding linkage tree. At the root is a V-fold mechanism linked to two parallel mechanisms and two extended patches.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

3.4 Parameter Space

The above ideas describe the possible connectivity and components of the pop-up. However, to fully determine a transforming pop-up, we must also need values for various geometric parameters, determining 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 the optimization algorithm in Section 4. We now consider how these parameters are defined in the ground plane.

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 12D (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 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. 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, and the dimension of the parameter space is reduced 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 total, a transforming pop-up is fully determined by its linkage tree, giving, its components and their connectivity, and the geometric parameters described above, giving their form.

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 state is a 2D pattern. Our main goal is to generate a pop-up, which appears similar to a user given *source pattern* in its untransformed state, and also appears similar to another 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 appearance of the shapes later). Secondly, any patches, or parts thereof, which are visible (i.e. not occluded by other patches) in both untransformed and trans-

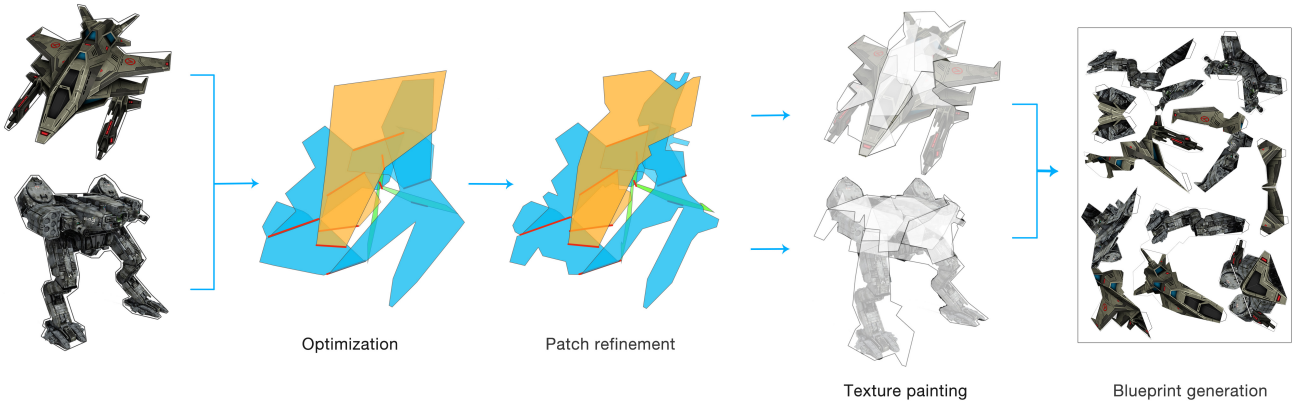


Fig. 10. Pipeline of our approach.

formed states, should flip over during transformation. If their front appears in the untransformed state, their reverse should appear in the transformed state. This allows the visible parts of the patches in untransformed and transformed states to be painted 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 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 iterating over all possible configurations of the linkage 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 restricts the pop-up to representing simple, approximate shapes. To allow 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 materials to the pop-up, the mechanism structures are preserved, as are the constraints—as long as we do not crop too much, as will be discussed in 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 instantiate 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 glueing them together.

We now consider these steps in further detail.

4.2 Optimization

4.2.1 *Energy function.* We define the optimisation 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]. Optionally, to avoid collision with the ground, we add a dummy ground patch of large enough area to enclose the design, and include it during collision checking. Users may choose whether to enable this option or not: if the goal is to place the pop-up inside a book, this option should be enabled, but if the goal is to make 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 is a weight to control 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 is defined in terms of a contour matching cost and an area matching cost:

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

The contour matching cost is computed using the shape context method [Belongie et al. 2002], and is given by:

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

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 computed

as:

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

where $\setminus$ is the set difference operator, $\|\cdot\|$ denotes the normalized area of a region, i.e. its area divided by the sum of the areas of all patches, and λ_a is a relative 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 later patch refinement step can crop them. However, unrepresented regions will show the ground patch as they will not be included in the transforming pop-up. Hence, in our implementation, we set $\lambda_a = 0.1$ to more strongly penalize unrepresented regions.

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 minimise regions which are visible in both untransformed and transformed states which do not show different sides of the paper. The flip term is simply defined as the sum of the areas of all regions which violate this constraint (again normalized as in Equation 4):

$$E_f = \sum_i \|R_i\|, \quad (5)$$

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 it can be modified to achieve greater performance.

Basic approach. To minimize the energy function defined in Equation 1 we iterate over all possible configurations for a linkage tree, up to a certain maximum size. To limit the number of configurations to reasonable bounds we restrict the maximal 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 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 pup-ups which are harder to construct with smaller pieces, and also which are 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. One patch of the mechanism connected to the pull-tab makes an angle with the pull-tab that varies smoothly from 0° to 180° as the mechanism operates. We can compute the angles at other hinges from simple geometric considerations. Since the energy function is highly non-linear, we use a genetic algorithm (GA) to optimize the geometric parameters. The fitness function for a configuration x is given by $-E(x)$. The parameters are each encoded using integers constrained to lie in an appropriate range. The initial population is generated at random. 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, and 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 to solve the above problem is time consuming, taking tens of hours to generate reasonable output. Our initial experiments showed that the GA after a few generations obtains an adequate global structure for the whole mechanism, but takes much longer to optimize in detail the vertex positions of each patch. Using this observation, we adopt 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 having the following properties:

—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 occluded by other patches in the other state.

We give an illustration of the greedy acceleration in Figure 11. Further speed is achieved by implementing the above algorithm using CUDA on GPU. Typically, the greedy strategy provides a 5–10$\times$ speed up, while GPU implementation provides approximately $10\times$ further speed up compared to a CPU based implementation. Overall, the final algorithm reduces the processing time from tens of hours to a few minutes.

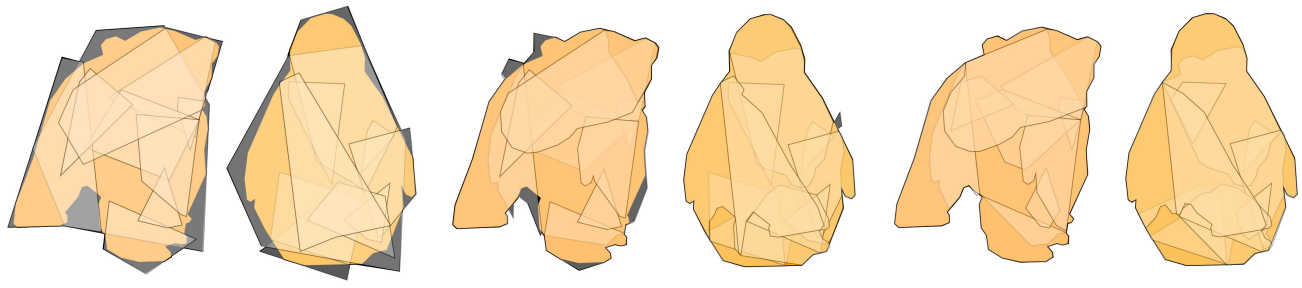
4.3 Patch refinement

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 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 such redundant regions as much as possible 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. However, we must also take care not to crop so much that we destroy the mechanism: hinges must not be completed cropped, cropping must not disconnect part of a patch from the rest of the pop-up, and cropping must not make any slider in a slide mechanism too short. In fact, in our experiments, we have not observed any of the above issues arising in practice.

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 process of transformation. Furthermore, 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 face by face; all faces need to be painted. For each face, we sequentially check:



(a) (b) (c)

Fig. 11. Greedy acceleration. (a) is an intermediate result by running brute force GA. By a patch refinement step we can get the result in (b). Note that some of the boundary vertices can be fixed in this step. Then by the proposed greedy acceleration strategy, we can get the final result in (c).

—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 only one texture (source or target pattern respectively).
—Otherwise (i.e. 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 the 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.

4.5 Blueprint generation

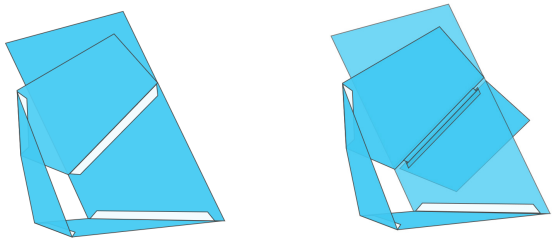
We finally generate a blueprint to help the user to make the transforming pop-up. A patch and any further polygons attached to it form an extended patch. See Figure 8(a): the two blue patches, and the two yellow patches, each form an extended patch. Both front and reverse of a given extended patch are printed on the same side of paper; the user folds it 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 it can be tricky to correctly register).

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. Hinges for *opposite* hinges (where planes cross) require a slit for manufacturing. How they are made is shown in Figure 12. The pull-tab is manually designed and is also printed in the blueprint. A typical blueprint is illustrated in Figure 13.

5. RESULTS

Our approach has been implemented in C++ on a PC with an Intel(R) 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.

Figure 14 shows three simple pop-ups generated by our system, transforming a disk into a triangle, a rectangle into a disk, and a rectangle into a triangle, respectively. The in-between shapes change in a reasonable way in these cases.



(a) (b)

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

Fig. 13. A blueprint.

Figure 15 shows two more complex examples, transforming a car into a robot, and an egg to a chick, respectively. Compared with the car to robot case in Reinhart's book, our result has a smaller area of redundant regions. The redundant regions in the egg to chick case are mainly by the chick's feet, which are thin and highly concave.

Figure 16 shows two examples with and without ground collision checking, transforming a caterpillar into a butterfly, and a penguin into a bear. For each case, both the two results give fascinating transformation.

Figure 17 shows two more examples transforming a hot air balloon into a space shuttle, and from the Great Sphinx of Giza into

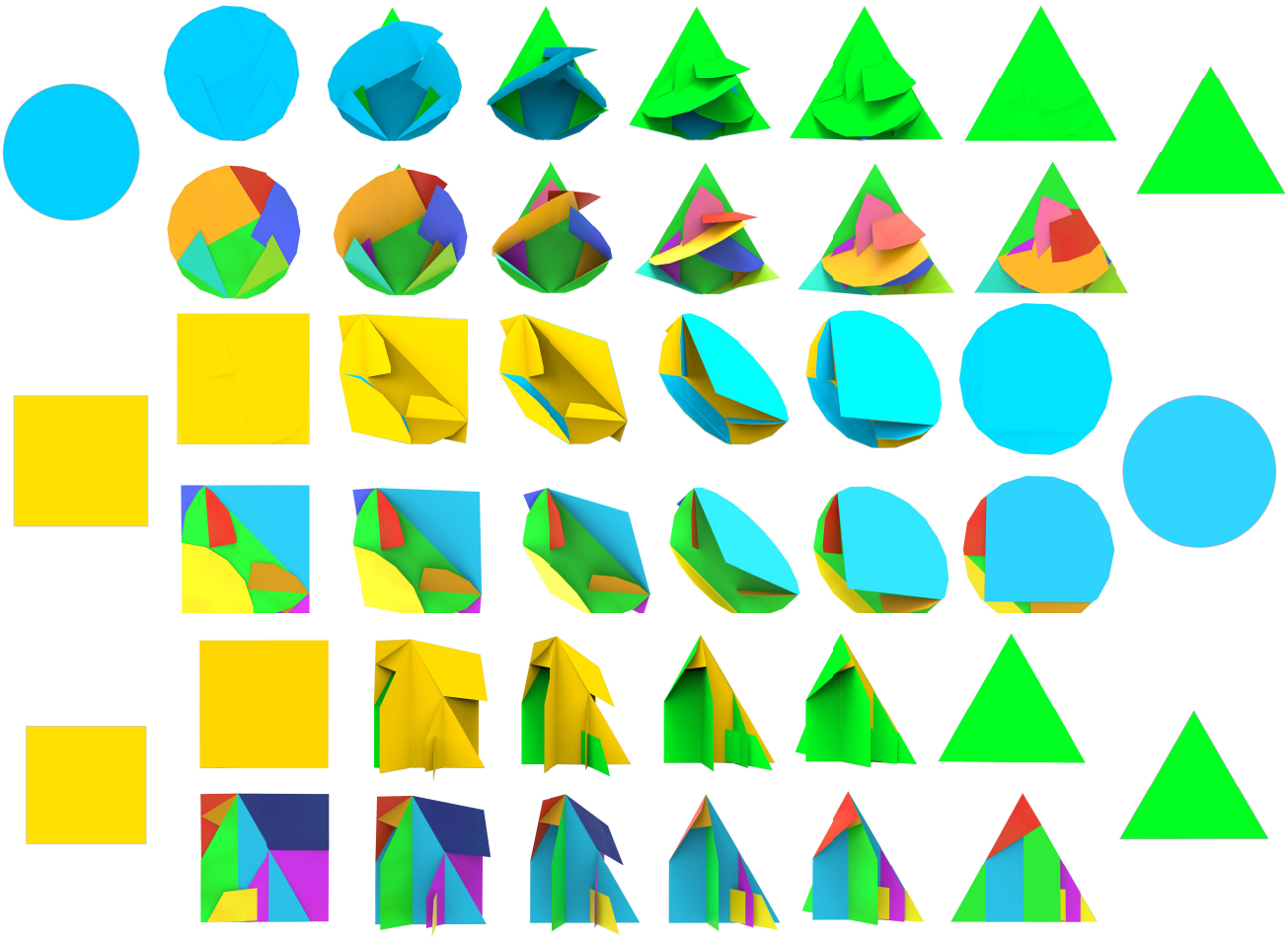


Fig. 14. Three simple examples, which from top to bottom, transform a disk into a triangle, a rectangle into a disk and a rectangle into a triangle. In each example, the upper row shows the appearance seen by the user, while the bottom row colours each patch differently, to visualize the mechanism.

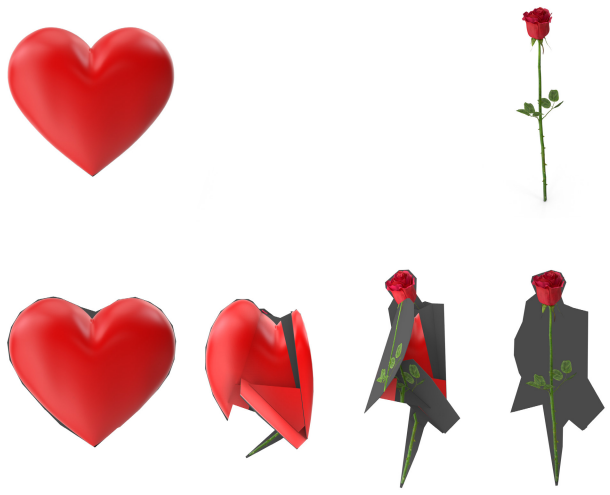


Fig. 18. Extreme example, with a very thin target pattern. Above: input patterns. Below: computed pop-up.

King Tutankhamun's Burial Mask. For each example, we show two different transforming pop-ups. Since a GA is a random algorithm, different runs can provide multiple solutions, allowing the user to generate alternative possibilities if the first attemot is not considered sufficiently aesthetically pleasing.

Finally, we also show an extreme case in Figure 18, which transforms a heart into a very thin rose. Since the two shapes differ two much, our approach inevitably produce large redundant regions.

We give statistics for each example in Table I, including the number of basic mechanisms used, and the computation time required both by the original GA algorithm and that of the GPU based greedy GA algorithm. The transforming animation of each example can be found in the supplementary video.

5.1 Evaluation: number of mechanisms

Our choice of using up to 5 mechanisms provides a good balance between quality and speed, as well as suitability of design for manufacture and robustness in use. Figure 19, we give the results with different number of mechanisms using the same input patterns, while Table II gives the numbers and kinds of mechanisms used in each case. The number of mechanisms of these results are presented in Table II. The result in Figure 19(a) uses only one mechanism

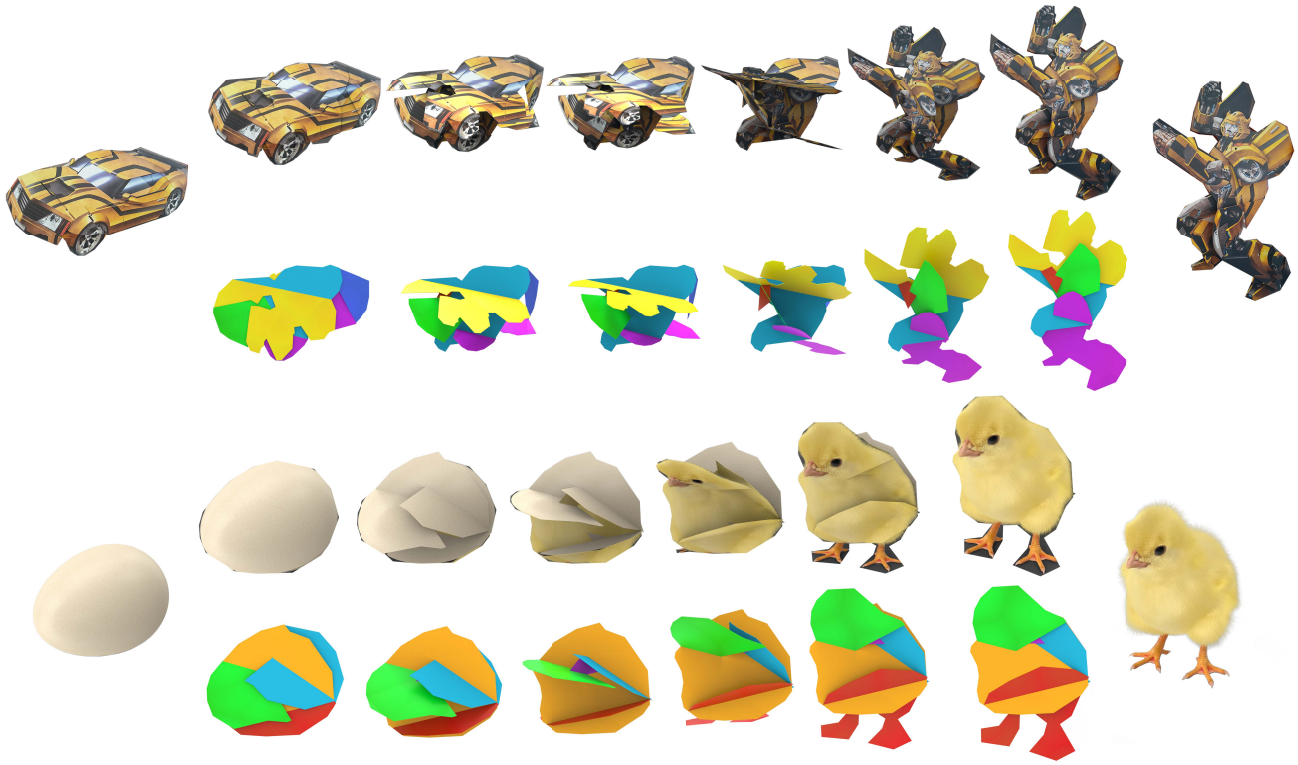


Fig. 15. Further examples, which transform a car into a robot, and an egg into a chick.

Table I. For each example: numbers of each kind of basic mechanisms, and extended patches, used—PM: parallel mechanism, VM: V-fold mechanism, SM: slide mechanism, EP: extended patch, and computation time. Examples computed using the basic algorithm have times given in minutes; those using the accelerated algorithm have times in hours.

Example Name	Figure	PM	VM	SM	EP	Time
Fighter–robot	2	1	2	0	12	16 hours
Disk–triangle	14	1	3	0	8	9 min
Square–disk	14	1	2	0	5	9 min
Square–triangle	14	1	2	1	5	5 min
Car–robot	15	1	2	0	6	11 min
Egg–chick	15	0	2	0	5	8 min
Caterpillar–butterfly 1	16	3	2	0	7	17 min
Caterpillar–butterfly 2	16	0	2	1	5	14 hours
Penguin–bear 1	16	1	3	0	8	11 min
Penguin–bear 2	16	0	3	0	7	13 hours
Balloon–shuttle 1	17	1	3	0	9	14 hours
Balloon–shuttle 2	17	1	3	0	8	10 min
Sphinx–mask 1	17	1	3	0	7	16 hours
Sphinx–mask 2	17	1	2	0	6	10 min

resulting in obvious and obtrusive redundant regions. The results in Figures 19(b) and 19(c) are much better, but still suffer from inaccurate boundaries. The results in Figure 19(d) and Figure 19(e) are both visually pleasing, but Figure 19(e) is more accurate than Figure 19(d). For more than 5 mechanisms, the computation time increases dramatically, and fabrication also becomes harder.

Table II. Numbers and types of mechanisms, and extended patches, used in each case in Fig. 19. PM: parallel mechanisms, VM: V-fold mechanisms, SM: slide mechanisms, EP: extended patches.

Subfigure	Number of Mechanisms	PM	VM	SM	EP
19 (a)	1	0	1	0	3
19 (b)	2	1	1	0	5
19 (c)	3	1	2	0	0
19 (d)	4	1	3	0	0
19 (e)	5	2	3	0	0

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 giving some intermediate states during optimisation from the caterpillar-to-butterfly example. Figure 20 shows various intermediate states, while the corresponding energies are given in Table III. Comparing Figure 20(a) with Figure 20(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 20(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 that the designer wishes to achieve, making the redundancy less noticeable.

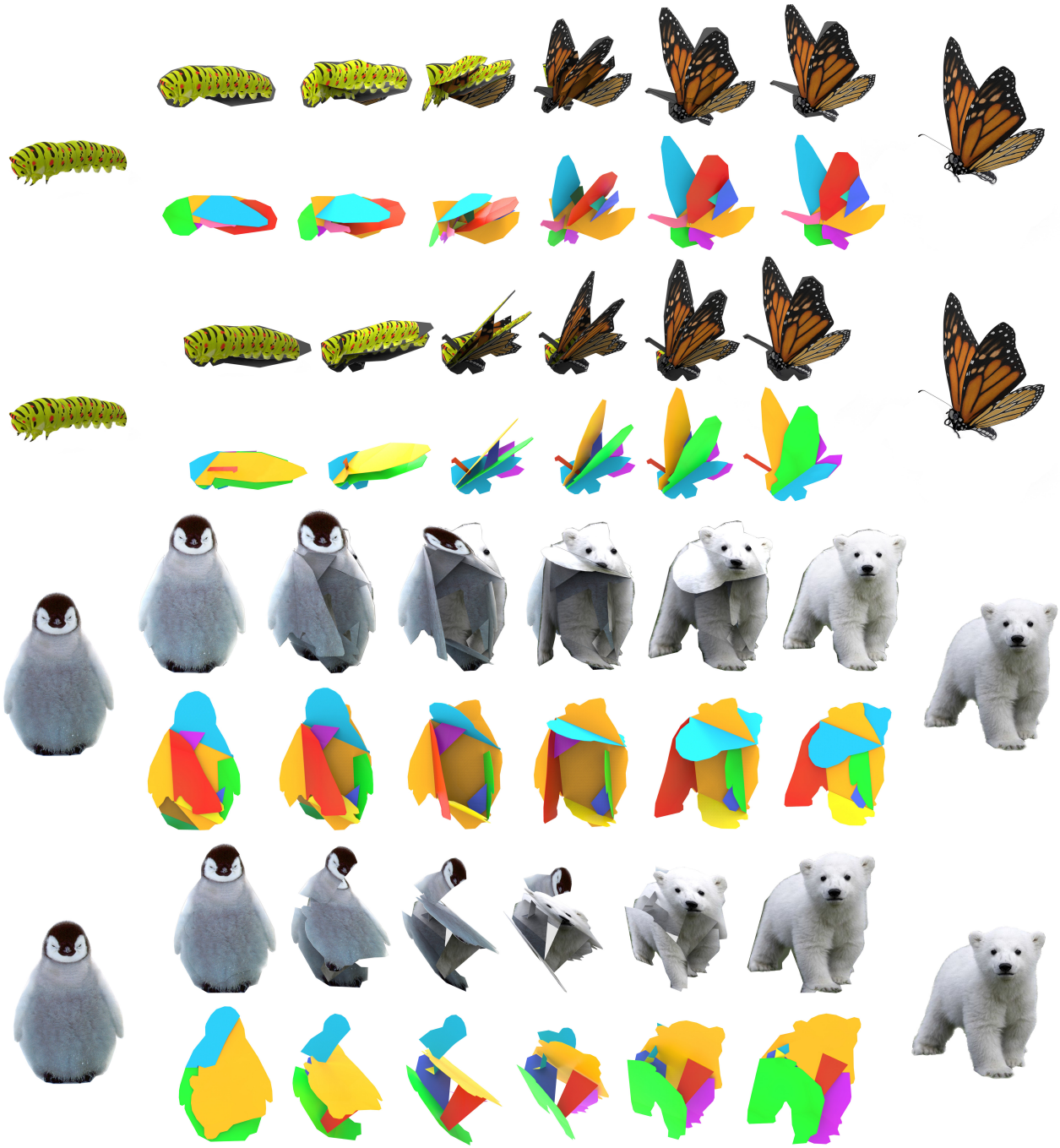


Fig. 16. Examples with (above) and without (below) ground collision checking, showing transformation of a caterpillar into a butterfly, and a penguin into a bear.

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 output is a blueprint for constructing the mechanism, so that the user can easily make a physical working mechanism. 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 can not readily work with shapes with curved boundaries. From the fabrication view, we have not considered how

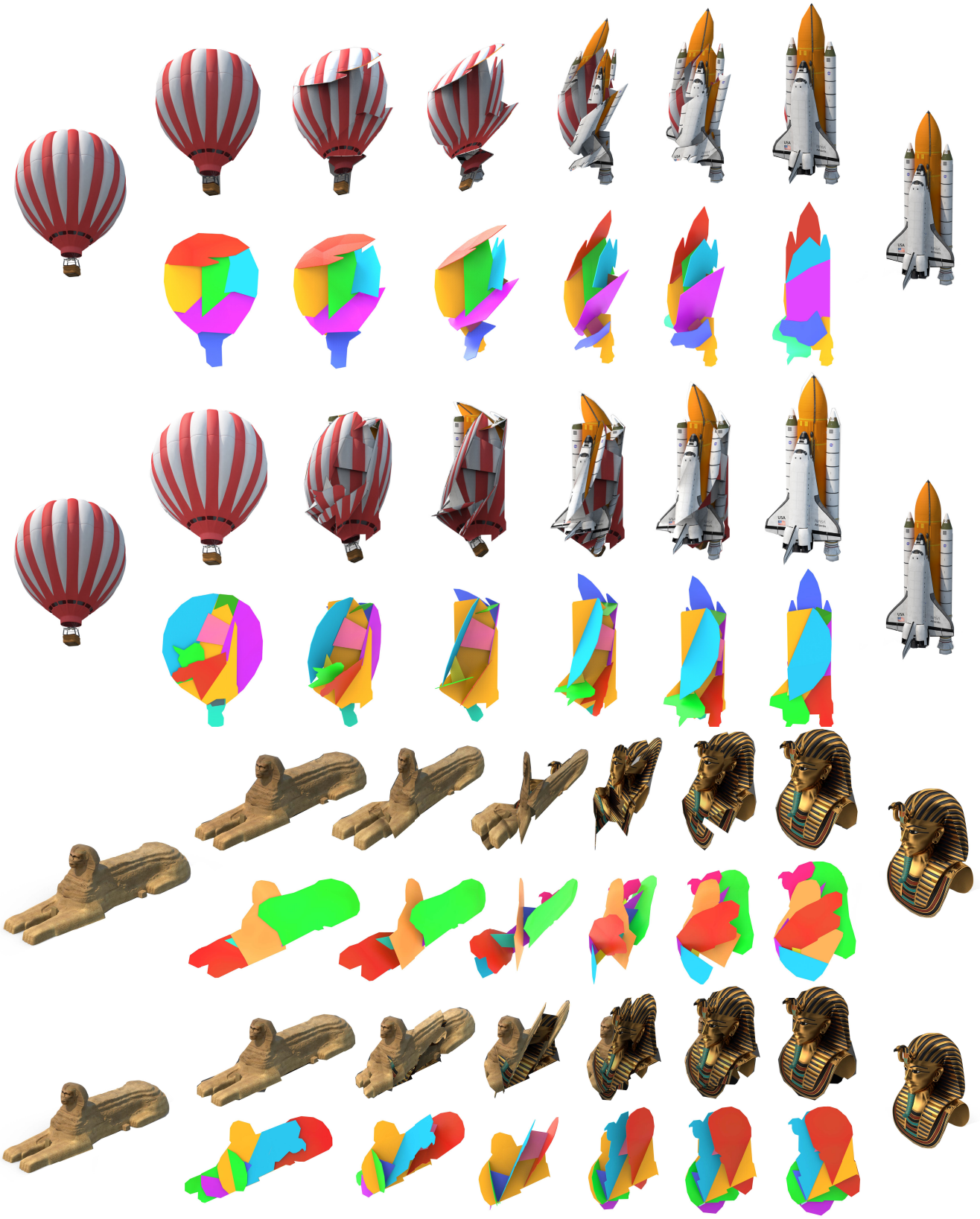
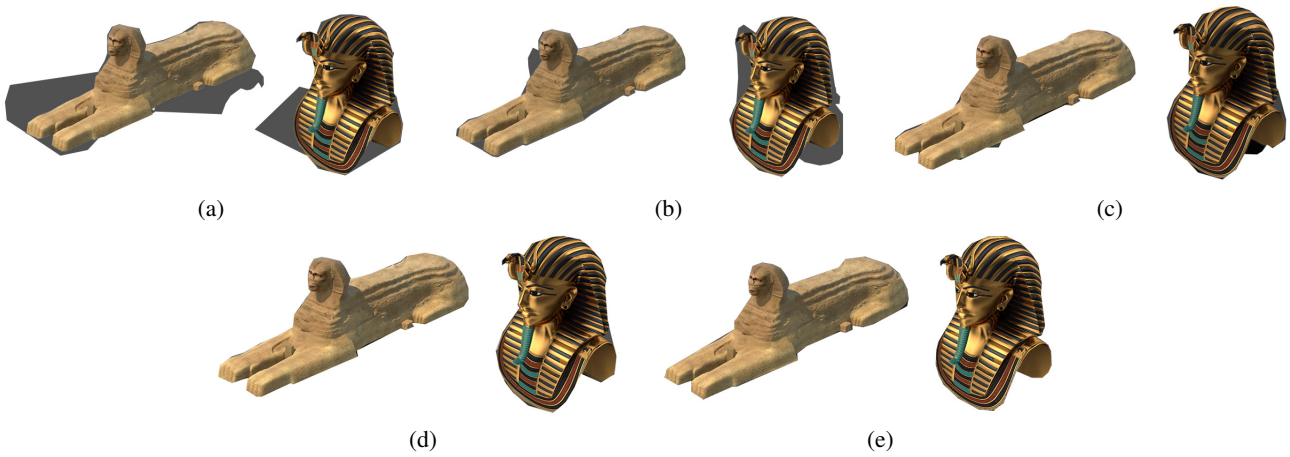


Fig. 17. Alternative solutions, showing two different transformations from a balloon into a space shuttle, and from the Great Sphinx of Giza into King Tutankhamun's Burial Mask.



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

Fig. 19. Results using varying numbers of mechanisms: from (a) to (e), 1 to 5 mechanisms.

(a) (b) (c)

Fig. 20. Intermediate results with different context matching cost and area matching cost (see Table III).

Table III. Statistics of shape context matching cost and area matching cost.

	Shape context cost	Area cost
(Fig. 20 (a))	0.1069	0.3424
(Fig. 20 (b))	0.1273	0.1507
(Fig. 20 (c))	0.0992	0.1575

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

APPENDIX

Degrees of Freedom of the Mechanisms Generated

In mechanics, the number of degrees of freedom (DoF) of a mechanism is the number of independent relative motions amongst a set of connected rigid bodies. In the following, we first prove that each of the three basic mechanisms we use have one DoF. Then, to prove that the whole popup mechanism has one DoF, we only need to prove that, using the allowable methods we have given, any combination of the basic mechanisms also has one DoF.

Theorem 1 (Kutzbach-Gruebler [McCarthy and Soh 2011]) The number M of degrees of freedom of a mechanism composed of n moving parts and j joints is given by:

$$M = 3(n-j) + \sum_{i=1}^{j} f_i,$$

where f_i denotes the number of degrees of freedom of the i-th joint.

Proposition 1 *A parallel mechanism composed of four quadrilateral patches T_i, $1 \leq i \leq 4$ connected cyclically by four mutually parallel hinges, and in which patches T_1 and T_3, and T_2 and T_4, are parallel in pairs has 1 DoF.*

Proof Such a parallel mechanism has 3 moving patches and 4 hinges. Each hinge has one DOF. Thus by Theorem 1, a parallel mechanism has $M_p = 3 \times (3-4) + 4 \times 1 = 1$ DoF.

Proposition 2 *A V-fold mechanism composed of four patches with four hinges, where the extended lines of the four hinges intersect at one point, and the angles α_{ij} between hinges h_i and h_j satisfy $\alpha_{14} = \alpha_{23}$ and $\alpha_{12} = \alpha_{34}$, has 1 DoF.*

Proof As for a parallel mechanism, a V-fold mechanism also has 3 moving patches and 4 hinges, each with 1 DoF. Thus a V-fold mechanism again has $M_v = 3 \times (3-4) + 4 \times 1 = 1$ DoF.

Proposition 3 *A slide mechanism composed of one rectangular patch (the slide patch), a hinge and a slit associated with a parallel mechanism or a V-fold mechanism (also constrained as defined in the paper) has 1 DoF.*

Proof In a slide mechanism, a virtual constraint between the slide patch and the slit, so it forms an RRPR linkage. Thus, it has 3 moving patches (apart from the slide patch) and each has 3 degrees of freedom [McCarthy and Soh 2011]. Thus a slide mechanism has $M_s = 3 \times (3-4) + 4 \times 1 = 1$ DoF.

Proposition 4 *Any mechanism which is the combination of these three basic mechanisms also has 1 DoF.*

Proof Let some combined mechanism containing i basic mechanisms have M^i DoF. Suppose we add one more basic mechanism to it, using the hinge linkage introduced in Section 3.3, giving a new mechanism with M^{i+1} DoF. Since all of the basic mechanisms are four-bar linkages, the way in which the newly added basic mechanism is combined with the existing mechanism is always the same. Altogether two patches and three hinges are added. According to Kutzbach-Gruebler's Theorem, the new mechanism has $M^{i+1} = M^i + \Delta M = M^i + 3 \times (2-3) + 3 \times 1 = M^i + 0 = M^i$ DoF. We must start with a single mechanism, and as noted, each of

the single mechanisms has 1 DoF. Thus, by induction, a mechanism formed from any number of basic mechanisms has 1 DoF.

REFERENCES

BELONGIE, S., MALIK, J., AND PUZICHA, J. 2002. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell. 24,* 4 (Apr.), 509–522.

COROS, S., THOMASZEWSKI, B., NORIS, G., SUEDA, S., FORBERG, M., SUMNER, R. W., MATUSIK, W., AND BICKEL, B. 2013. Computational design of mechanical characters. *ACM Trans. Graph. 32,* 4 (July), 83:1–83:12.

G. WINDER, BRIAN ANDP. MAGLEBY, S. AND L. HOWELL, L. 2009. Kinematic representations of pop-up paper mechanisms. *Journal of Mechanisms and Robotics 1,* 2.

GLASSNER, A. 2002a. Interactive pop-up card design, part 1. *IEEE Comput. Graph. Appl 20,* 1, 79–86.

GLASSNER, A. 2002b. Interactive pop-up card design, part 2. *IEEE Comput. Graph. Appl 20,* 2, 74–85.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of SIGGRAPH 96*. Annual Conference Series. 171–180.

JOSKOWICZ, L. AND SACKS, E. 1994. Configuration space computation for mechanism design. In *In Proceedings of IEEE International Conference on Robotics and Automation*. 1080–1087.

KOO, B., LI, W., YAO, J., AGRAWALA, M., AND MITRA, N. J. 2014. Creating works-like prototypes of mechanical objects. *ACM Trans. Graph. 33,* 6 (Nov.), 217:1–217:9.

LE, S. N., LEOW, S.-J., LE-NGUYEN, T.-V., RUIZ, C., AND LOW, K.-L. 2014. Surface and contour-preserving origamic architecture paper pop-ups. *IEEE Trans. Vis. Comput. Graph. 20,* 2 (Feb.), 276–288.

LE-NGUYEN, T., LOW, K., JR., C. R. R., AND LE, S. N. 2013. Automatic paper sliceform design from 3d solid models. *IEEE Trans. Vis. Comput. Graph. 19,* 11, 1795–1807.

LEE, Y., TOR, S., AND SOO, E. 1996. Mathematical modelling and simulation of pop-up books. *Computers & Graphics 20,* 1, 21 – 31.

LI, H., HU, R., ALHASHIM, I., AND ZHANG, H. 2015. Foldabilizing furniture. *ACM Trans. Graph. 34,* 4 (July), 90:1–90:12.

LI, X.-Y., JU, T., GU, Y., AND HU, S.-M. 2011. A geometric study of v-style pop-ups: Theories and algorithms. *ACM Trans. Graph. 30,* 4, 98:1–10.

LI, X.-Y., SHEN, C.-H., HUANG, S.-S., JU, T., AND HU, S.-M. 2010. Popup: automatic paper architectures from 3d models. *ACM Trans. Graph. 29,* 4, 111:1–9.

MCCARTHY, J. M. AND SOH, G. S. 2011. *Geometric Design of Linkages*. Vol. 11. Springer-Verlag New York.

MCCRAE, J., SINGH, K., AND MITRA, N. J. 2011. Slices: A shape-proxy based on planar sections. *ACM Trans. Graph. 30,* 6 (Dec.), 168:1–168:12.

MITANI, J. AND SUZUKI, H. 2003. Computer aided design for 180-degree flat fold origamic architecture with lattice-type cross sections. *Journal of Graphic Science of Japan 37,* 3, 3–8.

MITANI, J. AND SUZUKI, H. 2004. Computer aided design for origamic architecture models with polygonal representation. In *Proceedings of the Computer Graphics International*. 93–99.

MITANI, J., SUZUKI, H., AND UNO, H. 2003. Computer aided design for origamic architecture models with voxel data structure. *Transactions of Information Processing Society of Japan 44,* 5, 1372–1379.

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

OKAMURA, S. AND IGARASHI, T. 2009. An interface for assisting the design and production of pop-up card. *Lecture Notes in Computer Science 5531,* 2, 68–78.

REINHART, M. 2013. *Transformers: The Ultimate Pop-up Universe*. LB Kids; Pop edition.

RUIZ, C. R., LE, S. N., YU, J., AND LOW, K.-L. 2014. Multi-style paper pop-up designs from 3d models. *Computer Graphics Forum 33,* 2, 487–496.

SCHWARTZBURG, Y. AND PAULY, M. 2013. Fabrication-aware design with intersecting planar pieces. *Comput. Graph. Forum 32,* 2, 317–326.

WON, J. AND LEE, J. 2016. Shadow theatre: Discovering human motion from a sequence of silhouettes. *ACM Trans. Graph. 35,* 4 (July), 147:1–147:12.

XIN, S., LAI, C.-F., FU, C.-W., WONG, T.-T., HE, Y., AND COHEN-OR, D. 2011. Making burr puzzles from 3d models. *ACM Trans. Graph. 30,* 4 (July), 97:1–97:8.

ZHOU, Y., SUEDA, S., MATUSIK, W., AND SHAMIR, A. 2014. Boxelization: Folding 3d objects into boxes. *ACM Trans. Graph. 33,* 4 (July), 71:1–71:8.

ZHU, L., XU, W., SNYDER, J., LIU, Y., WANG, G., AND GUO, B. 2012. Motion-guided mechanical toy modeling. *ACM Trans. Graph. 31,* 6 (Nov.), 127:1–127:10.