Image-based Popup Craft Design via MIP Optimization

Chen Liu*

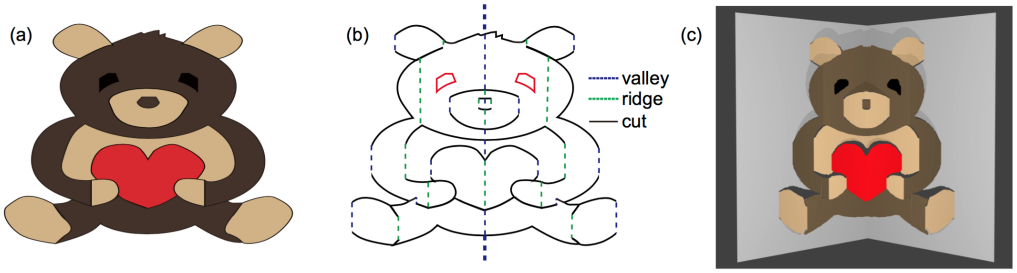

Figure 1: *(a) Input 2D shape and its segmentation. (b) Generated cuts and folds. (c) 3D pop-up rendering.*

Abstract

In this project, we focus on automatic popup craft design with the problem setting from [Liu et al. 2014]. But instead of using the greedy method used in [Liu et al. 2014], we use non-linear optimization to optimize the design globally. The optimization formulation is designed such that an optimal solution is a popup design which 1) follows the input shape, 2) is foldable, and 3) is connected, and 4) is stable (See Section 1).

1 Introduction

Popup craft, introduced by Masahiro Chatani in 1980, is a design of cuts and folds on a single piece of paper. Complex and interesting structure pops up when the paper is opened. The special structure of a popup craft makes it appealing but hard to design. In this work, we seek an automatic design method via an optimization process.

Given either a Vector image or a scalar image, we first segment it to get a segmentation image $\mathcal{S}$ which stores a segment index for each pixel. Based on $\mathcal{S}$, we find initial fold lines between pair of different segments. Given these initial fold lines, we find other fold line candidates in a sense that a new fold line candidate affects the topology of initial fold lines. With all the fold lines as candidates, we determine their activeness, convexity, and positions in an optimization framework. By adding auxiliary variables and constraints, we enforce an optimal solution to be a popup craft design with satisfactory properties. First, a popup craft must be foldable, which means it can be opened and folded without breaking its geometry (see [Li et al. 2010] for more details). Second, a popup craft is more appealing when it is stable, which means the structure is stable when two border of the paper is holded (see [Li et al. 2010] for more details). Third, because the number of segments of the popup craft is limited in our image-based problem setting, the connectivity, which enforces there is only one connected structure in the popup craft, becomes important. Fourth, the popup craft should follow the input shape.

2 Related Work

Due to rigid paper crafting constraints, the OA design process is often time consum- ing and requires considerable skills. Several computer-aided design tools have been developed to provide a virtual design environment and assist the design process (see [**?**] and the references therein). However, the ultimate placement of cuts and folds still depends on the user, posing the design process trouble- some and highly skill-demanding. To further simplify OA design, Li et al. [Li et al. 2010] proposed a fully automatic algorithm to convert building models to paper architectures. Le et al.[Le et al. 2014] presented a surface-and-contour-preserving method to pop up 3D models with more freeform shapes. Liu et al.[Liu et al. 2014] proposes an image-based approach but the greedy approach fails in many cases.

3 Popup Graph

Given the input image, either a vector image or a scalar image, we first segment the image. For a vector image, we read the segment information from the image directly. For a scalar image, we segment the image using Watershed algorithm. With the image segmentation, we then extract a family of popup graphs, each of which defines a specific popup design with unique configuration of fold lines and cuts. The common attribute of the family is derived from the input segmentation, and our goal to choose the best one from the family based on appealing popup properties. In this section, we will define popup graph, its family and how we build such a family from the input segmentation.

From the example in Figure **??** we can see, a popup design contains fold lines, cuts and patches. Multiple patches consist of an input segment and they are separately by fold lines. Cuts and fold lines form segment boundaries. So once the fold lines are given, we can acquire patch information by cutting segments with fold lines and retrieve cut information by looking at segment boundary. So with input segmentation, fold lines are the only information we need to define a popup design. A popup graph is thus defined by fold line properties.

Fold lines are vertical lines on the image domain, which either connect two segments or cut a segment through. Use (u, v) to represent the image coordinate, two end points of a fold line f are $(u(f), v_1(f))$ and $(u(f), v_2(f))$. So $u(f)$ is a key attribute we want to optimize. But instead of optimizing $v_1(f)$ and $v_2(f)$, we seek to pre-define a unique mapping between $u(f)$ and $(v_1(f), v_2(f))$ pair for each fold line so that, we can read out $v_1(f)$ and $v_2(f)$ based on $u(f)$. The unique mapping property holds once we specify a range for a fold line (see Figure **??**). To ensure the completeness of solution space, we find all possible fold lines and introduce a variable $a(f)$ for fold line f to indicate whether it is active or not. Besides $u(f)$ and $a(f)$, we add $c(f)$ to indicate the convexity of a fold line (whether it points outwards or in-

*chenliu@wustl.edu

wards). We will explain how to find all fold line candidates and define their image-specific attributes including the unique mapping between $u(f)$ and $(v_1(f), v_2(f))$ pair. A popup graph G is a graph of fold lines with a specific configuration of $u(f)$, $a(f)$, and $c(f)$ together with image-specific attributes. A popup graph family $\mathcal{G}$ is a family of popup graphs with common image-specific attributes (i.e. popup graphs derived from the same image).

3.1 Intersection Fold Lines Candidates

We call a fold line which connects two segments as an "intersection fold line". For any given pair of neighboring segments, there could be an arbitrary number of fold lines with arbitrary position along the intersection. Ideally, we want one fold line candidate for each possible position, but then the optimization becomes intractable. In practice, we first divide the intersection into multiple parts such that each part is supposed to contain only one fold line and there exists unique mapping between $u(f)$ and $(v_1(f), v_2(f))$ pair. For this purpose, we define a *slope* as a part of the intersection where x is monotonically increasing or decreasing over y as shown in Figure **??**. Then we assign one fold line candidate for each *slope*. Note that the fold line can appear at any location at the slope, but we use the location with the highest feasibility score as the default location for the fold line.

To define the feasibility score for a fold line candidate at given position, we look at a local window around that position. The size of the window has clear physical meanings. When we make actual popup crafts out of a hard paper, a fold line has to be long enough so that it connects two patch stably. Also, we want patches beside it to be wide enough so that the fold line is easy to fold by hand. The height of the local window is chosen to be the minimal fold line length and the width of the local window is chosen to be twice of the minimal patch width beside a fold line. For a fold line f to appear at pixel p, denote its left segment as s_l and its right segment as s_r, the score of the fold line f, is determined as 1.

$$S(f, p, s_l, s_r) = \frac{|W_l(p) \cap R(s_l)| * |W_r(p) \cap R(s_r)|}{(0.5H^f W^f)^2} \quad (1)$$

3.2 Find Region Fold Line Candidates

Intersection fold lines connect segments and make the popup craft more appealing and stable, but often it is necessary to add fold lines inside segments to make the popup craft foldable. We call these fold lines "split fold lines". As a split fold line splits a segment into halves, it also splits intersection fold lines of this segment into two groups. In this sense, a split fold line changes the topology of intersection fold lines. Appearing at different locations inside the segment, a split fold line changes the topology in different ways. In this manner, we divide the segment into multiple regions, each of which contains a split fold line candidate.

3.3 Structure and Properties

A popup graph has fold line candidates as nodes, and their neighboring relations as edges. Two fold lines are regarded as neighbors only when they belong to the same segment and there is no fold line between them. A popup graph has the following properties:

Background Enclosing: The background segment has one split fold line candidate called *background fold line* which is always active. The *background fold line* divides the background segment into two halves. We call these two halves *left background patch* and *right background patch*. This corresponds to the two halves of most blessing cards. We add two split fold lines at left and right image border to simplify notation. Then the left image border fold line is the source of the graph and the right image border fold line is the sink of the graph.

Directed Acyclic Graph (DAG): The left-right orientation between fold line candidate pairs is well-defined and the the graph excluding the background fold line is also a DAG.

4 Optimization

Given a popup graph family, we need to determine the following three sets of variables in order to choose the optimal popup graph.

1. $a(f)$: A binary variable indicating the activeness of fold line f.
2. $c(f)$: A binary variable indicating the convexity of fold line f (whether the fold line points outwards ($c(f) = 1$) or inwards ($c(f) = 0$).
3. $p(f)$: A integer variable indicating the position of fold line f.

The goal is to determine these variables so that the popup graph satisfies four key properties: foldability, connectivity, stability and consistency. Additional properties can be enforced according to user input. We can view our task as an optimization problem $G^{OPT} = optimize(\mathcal{G})$ which takes a popup graph set $\mathcal{G}$ as input and outputs the optimal popup graph:

$$\begin{aligned} G^{OPT} = & argmax_G \quad Consistency(G) \\ & s.t. G \in \mathcal{G} \quad and \quad Foldability(G) = TRUE \quad and \quad Connectivity \end{aligned} \quad (2)$$

We formulate the optimization problem as a mxied integer programming problme. We will explain how to formulate each property as either quadratic objective terms or quadratic constraint terms and provide heuristics for making the optimization problem tractable.

4.1 Foldability

Foldability Definition:

Variable Definition: We associate a set of variables $c(f)$ determining the convexity of f, that is . A set of variables $X(f)$ determining the X coordinate of f, and $Y(f)$ determining the Y coordinate of f.

There is $x(f) + y(f) = p_u(f)$ ($p(f) = (p_u(f), p_v(f))$).

The orientation constraints involve $c(f)$. For fold line neighbor pair (f_l, f_r), there is:

$$\begin{aligned} c(f_l) &= 1 - c(f_r) \quad \text{if } a(f_r) = 1 \\ c(f_l) &= c(f_r) \quad \text{otherwise} \end{aligned} \quad (3)$$

The position constraints is formed as:

$$\begin{aligned} x(f_l) < x(f_r) \quad & y(f_l) = y(f_r) \text{ if } c(f_r) = 1 \\ x(f_l) = x(f_r) \quad & y(f_l) < y(f_r) \text{ otherwise} \end{aligned} \quad (4)$$

4.2 connectivity

Due to the fact that the number of segments is small in our image-based design process, we prefer one intriguing structure containing all segments instead of multiple separated simple structures.. For

this reason, we enforce two type connectivity here. First, each patch should have at least one path from both left background patch and right background patch. Second, after taking background patches away, the rest graph should have only one connected component. The connectivity property is incorporated into the optimization formulation as follows.

As new fold lines will never be cut, the connectivity property is considered based on initial patches. The first type of connectivity is enforced by simply adding a constraint that each initial patch (except the background patch) has at least one active left initial fold line and one active right initial fold line. The proof is trivial as the graph has non-loop property. We re-phrase the second connectivity constraint as there exists at least one connection between any pair of patches (background patches are excluded). Here the connection exists when:

1. For a pair of neighboring patches, at least one fold line between them is active.
2. For a pair of non-neighboring patches, they both have connection with at least one other patch.

Due to the symmetric definition of connection, we can simplify the constraint as there exists at least one connection between each patch and one denoted patch s. The denoted patch is a randomly picked non-background patch. Then we define the connection depth for each patch as the length of the shortest path to patch s. We use binary variable c_{pd} to indicate whether patch p has connection depth d ($d \in [1, MAX\_DEPTH]$). Based on c_{pd}, the connectivity constraint is formulated as $\sum_d c_{pd} = 1$. According to the definition of connection, c_{pd} subjects to:

$$\begin{cases} c_{p1} \leq \sum_{f \in F(p,s)} a(f) & \text{If patch } p \text{ is a neighbor} \\ c_{p1} = 0 & \text{If patch } p \text{ is not a neigh} \\ c_{pd} <= \sum_{q \in N(p)} (c_{q(d-1)} \sum_{f \in F(p,q)} a(f)) & \end{cases}$$

Note that some patches do not have active fold lines because they lie inside another patch (see the eyes in the bear example). We call them "island patches". For island patches, connectivity no longer holds. For this reason, we add another set of binary variables $i(p)$ to indicate whether a patch is an island patch or not. We modify the above connectivity constraints so that they are disabled for island patches. And an island patch has no fold line, so there should be $a(f) <= 1 - i(p) \quad \forall f \in F(p)$.

4.3 Stability

We associate a set of binary variables $s(f)(d)$ indicating the stability of f at depth d. Here the depth can be interpolated as the following procedure. Lf, Rf, and all fold lines on the background patches are stable at depth 0. Now if we add two patches between background patches forming a stable structure, the corresponding fold line will be stable at depth 1. We can continue to add stable structure based on current structure and the depth increases as the procedure goes on.

We use the idea of forming the popup by iteratively adding stable structure to formulate the stability constraint. That is, we check a building block at depth d to see whether it is stable or not based on previous building blocks (with stability depth less than d).

We consider five types of stable structure:

1. A fold line is stable if it is on the same patch with two known stable fold lines. ([Li et al. 2010])
2. A fold line is stable if it is on the same patch with one known stable fold line and on another patch with another known stable fold line. ([Li et al. 2010])
3. A fold line is on a B-path or F-path. ([Le et al. 2014])
4. A fold line is double connected with two stable patches (ours).
5. A fold line is double connected with a stable patch and directly connect with a stable patch (or a patch which is double connected with a stable patch) (ours).

These cases are illustrated in figure 4.

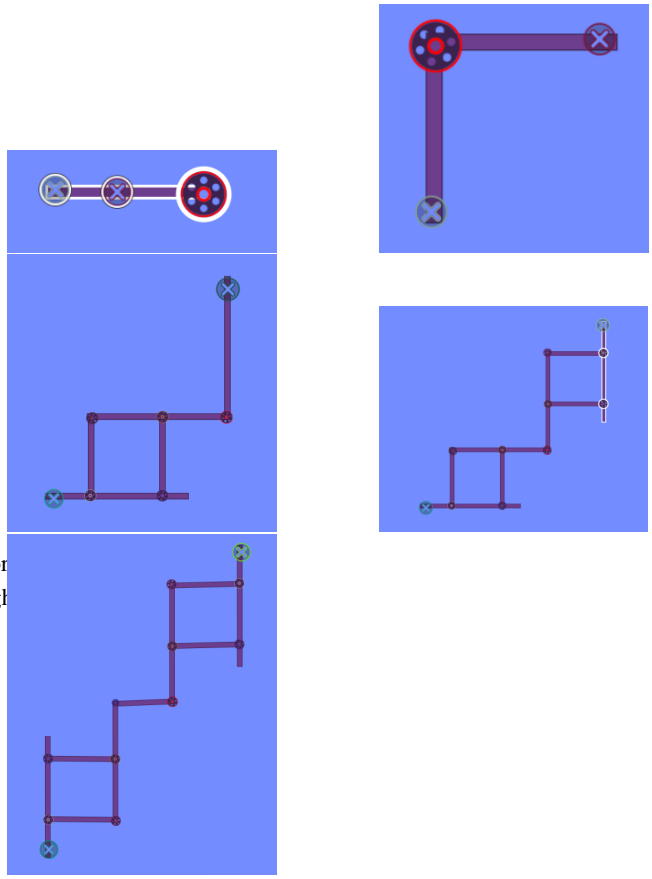

Figure 2: *The anchor with X sign represents a stable fold line and the anchor with red color is the fold line to be considered*

The process of determining stability is as follows:

1. Background fold lines are stable with stability depth 0.
2. For stability depth k, we find either the above structures based on the stable fold lines with stability depth less than k, and mark fold line in such structures as stable with stability k.
3. Repeat this process until all fold lines are marked as stable at some point. (We set a stability depth limit in practice.)

4.4 Consistency

As we design the popup craft based on the input image, we want the design result to follow the input shape. Some geometry changes are avoidable since we need to add fold lines between patches, but still we want to minimize the geometry change. Although more complicated metric could be applied here, we use a simple metric

to measure the geometry change. That is, the geometry change for a fold line is the amount of shift from its initial location to its final location, and the geometry change for the design result is the summation of fold line geometry changes. So the consistency is measured as: -$\lambda_{pos} \sum_f (pos(f) - pos_i(f))^2$.

4.5 Heuristics

Optimization **??** gives us optimized popup graph which is foldable, connected, stable and consistent with the input image. However, the optimization problem is intractable to any existing solvers in many cases due to the large number of variables and constraints (most of which are brought for the stability constraint). So instead of optimization **??** directly, we decouple the optimization process to decrease the complexity by the following two heuristics:

Stability Constraint Decoupling: The stability constraint brings in $(\#f)^3$ auxilliary variables and constraints which cause the most difficulty. Our approach is to ignore stability while optimizing other terms and then check whether the returned solution satisfies the stability constraint. If the solution is not stable, we conduct the optimization again while excluding the unstable solution. We repeat the process until we find a stable solution. To find a stable solution with fewer such attempts, we guide the process by adding another objective term. Our observation is that a good popup graph design has two attributes: 1) most intersection fold lines are active as they add connections between segments which make the popup graph more intriguing and stable, and 2) a region fold line becomes active only when it is essential to make the popup graph foldable. We use f_i to denote an intersection fold line and f_r a region fold line, then the objective term is $max(\sum a(f_i) - \sum a(f_r))$. This term is associated with an arbitrary large weight so that it has higher priority than the consistency objective. We denote this variation of $G = optimize^t(\mathcal{G})$ as $G = optimize^t(\mathcal{G})$ indicating only topology is considered without the stability constraint.

Child Segment: A child segment a segment which lies on the same plane with its parent segment and has no fold line (like the eye of the bear). Child-parent segment pair happens when two segments have different textures but are supposed to be on the same plane. A child segment has no active fold line and thus is neither connected, or stable. So we ignore child segments when optimizing the rest graph and add them back after the optimization. For this purpose, we detect segments which are enclosed by another segment. We treat them as child segments and ignore them in the optimization. Note that, such detected segments are not always real child patches (like the nose of the bear). We address this issue by simply conduct the optimization with the rest part of the graph fixed. This time, we only consider foldability constraint. We call this optimization $optimize^c(\mathcal{G})$. Users can also denote other child patches.

Putting these two heuristics together, we reach algorithm **??**.

5 Results

Current results delete some island patches like eyes for both cases. The island patches can be added back after the rest part is optimized. We haven't done that part yet.

6 Conclusion

To our knowledge, this is the first work to design image-based popup craft via optimization.

Algorithm 1: Popup Graph Optimization

input : A scalar image or vector image I
output: A foldable, connected and stable graph design which is consistent with the input image or a failure flag.

Initialization: $\mathcal{G} = buildPopupGraphSet(I)$
$\mathcal{G}' = \{G | G \in \mathcal{G} \quad and \quad$ G has no child patch$\}$;
$max\_iters = 10, iter = 0$;
while *true* **do**
  $G^t = optimize^t(\mathcal{G}')$;
  if $stable(G^t)$ **then**
    break;
  end
  $\mathcal{G}' = \mathcal{G}' \, G^t$;
  $iter = iter + 1$;
  if $iter > max\_iters$ **then**
    return *FAILURE*;
  end
end
$\mathcal{G}^t = \{G \approx G^t | G \in \mathcal{G}\}$;
$G^{OPT} = optimize^c(\mathcal{G}^t)$;
return G^{OPT};

References

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. *Visualization and Computer Graphics, IEEE Transactions on 20*, 2, 276–288.

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.

LIU, C., YANG, Y.-L., LEE, Y.-H., AND CHU, H.-K. 2014. Image-based paper pop-up design. In *ACM SIGGRAPH 2014 Posters*, ACM, 36.

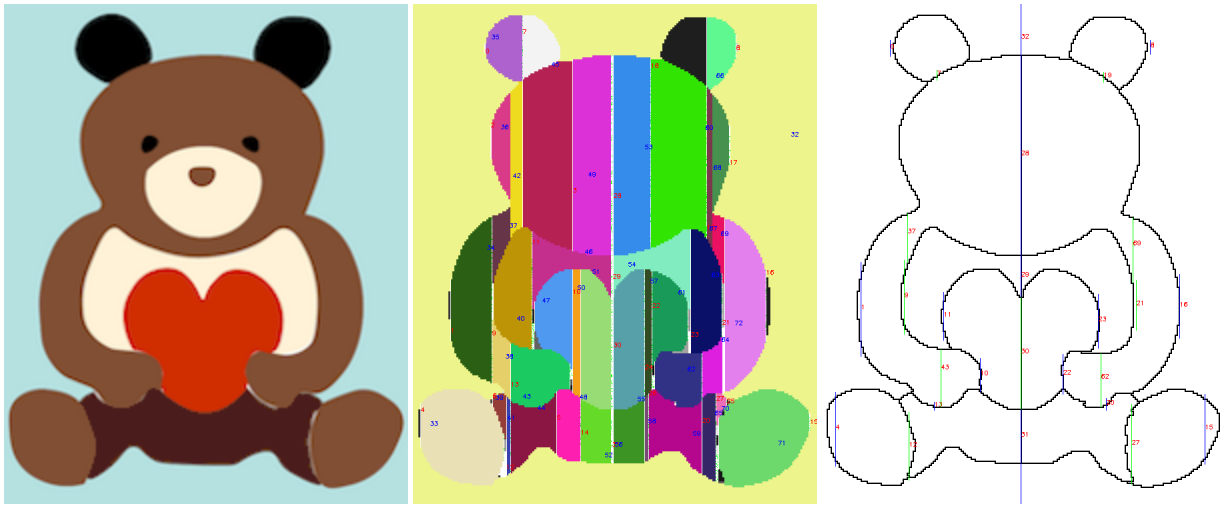

Figure 3: *Bear example: (a) input shape (b) fold line candidates (c) popup design*

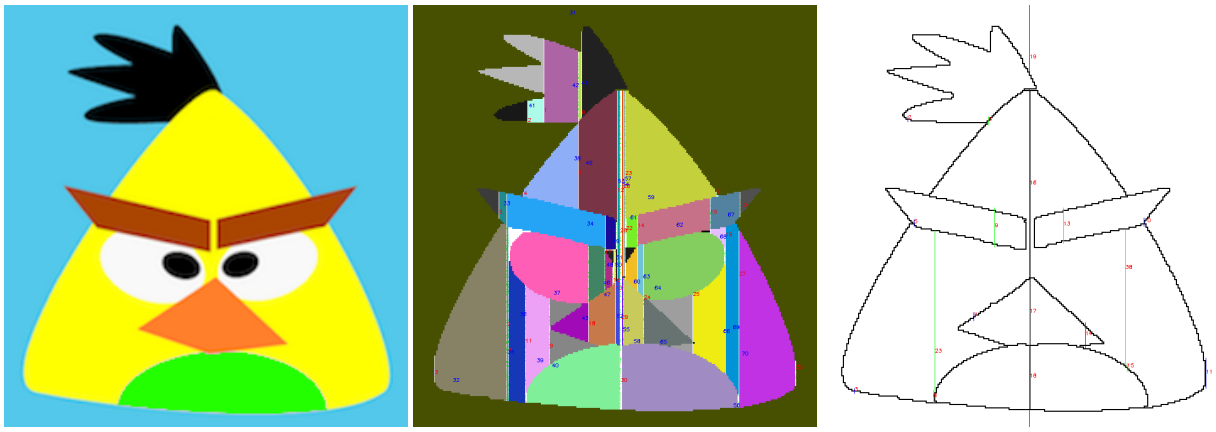

Figure 4: *Angrybird example: (a) input shape (b) fold line candidates (c) popup design*