Automatic Paper Sliceform Design from 3D Solid Models

Tuong-Vu Le-Nguyen, Kok-Lim Low, Conrado Ruiz Jr., and Sang N. Le

Abstract—A paper sliceform or lattice-style pop-up is a form of papercraft that uses two sets of parallel paper patches slotted together to make a foldable structure. The structure can be folded flat, as well as fully opened (popped-up) to make the two sets of patches orthogonal to each other. Automatic design of paper sliceforms is still not supported by existing computational models and remains a challenge. We propose novel geometric formulations of valid paper sliceform designs that consider the stability, flat-foldability and physical realizability of the designs. Based on a set of sufficient construction conditions, we also present an automatic algorithm for generating valid sliceform designs that closely depict the given 3D solid models. By approximating the input models using a set of generalized cylinders, our method significantly reduces the search space for stable and flat-foldable sliceforms. To ensure the physical realizability of the designs, the algorithm automatically generates slots or slits on the patches such that no two cycles embedded in two different patches are interlocking each other. This guarantees local pairwise assembility between patches, which is empirically shown to lead to global assembility. Our method has been demonstrated on a number of example models, and the output designs have been successfully made into real paper sliceforms.

Index Terms—paper sliceform, lattice pop-up, paper scaffold, papercraft, computer art, shape abstraction.

1 Introduction

Paper sliceforms, also known as *lattice-style paper pop-ups*, is a popular and frequently used techniques in paper pop-ups books and cards. A paper sliceform is a 3D structure consisting of multiple planar patches of paper. These patches usually depict the cross-sections of a solid object, taken along two orthogonal directions. Thin straight slots or slits are cut on these patches along their intersections such that the patches can be slotted together, and the intersections act as hinges that allow the paper sliceform to be opened (popped-up) or folded flat (no folding of patches actually takes place since every pair of intersecting patches rotate with respect to each other about their intersection). From these sets of orthogonal patches very intricate sliceforms can be made. Examples of sliceforms are shown in Fig. 2.

Software tools such as Autodesk 123D Make and SliceModeler for Google Sketchup have been available for people to generate paper sliceform designs from solid models [1], [2]. However, our experience with these systems has found that making a valid sliceform design can be difficult and often impossible. A common limitation of these systems is that they do not guarantee that the final design is stable and it can be physically assembled without the need to break some patches, such as the unstable and interlocking rings example shown in Fig. 3.

• *T.-V. Le-Nguyen, K.-L. Low, C. Ruiz Jr. and S. N. Le are with the Dept. of Comp. Science, National University of Singapore, Singapore E-mail: tuongvu@gmail.com, {lowkl,conrado,lnsang}@comp.nus.edu.sg*

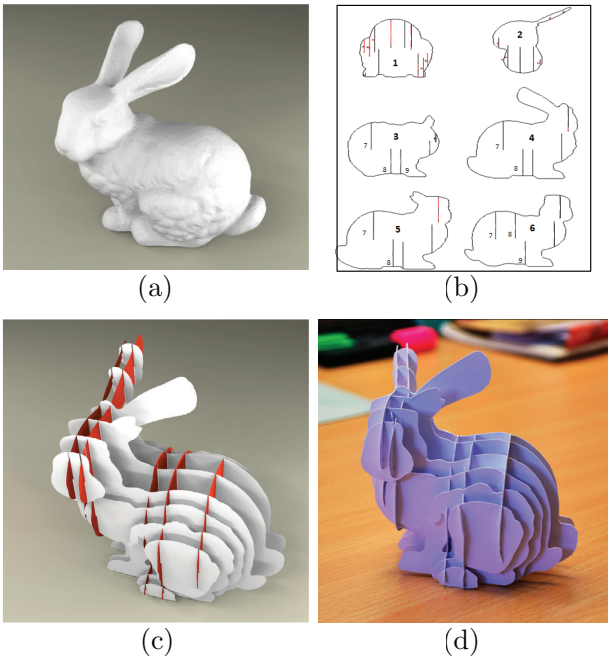

Fig. 1. (a) An input 3D model (b) One of the 2D layout pages generated by our algorithm (c) Rendering of the synthetic paper sliceform (d) A real hand-made paper sliceform assembled from the 2D layout.

We have developed an automatic algorithm that generates paper sliceform designs from 3D solid models. The designs are guaranteed to be stable and empirically validated to be physically realizable. Given an input 3D solid model (represented as polygon meshes), a sliceform design is generated by our system as a set of 2D patches printed on one or more pages. Each patch is indicated by its outlines, and

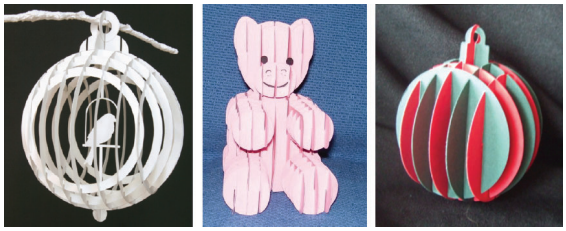

Fig. 2. Some examples of real paper sliceforms.

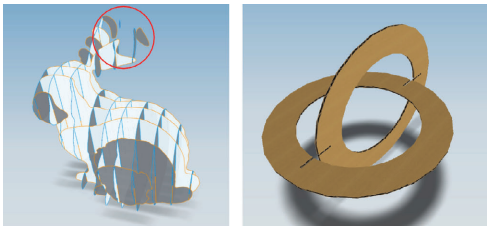

Fig. 3. Sample Autodesk 123D Make models. (Left) Unstable patches—some patches of the bunny's ears are not connected. (Right) Unrealizable—the two interlocking rings cannot be physically assembled.

straight line segments are drawn to indicate slots. Each patch is labeled with a unique number, and each slot line is labeled with the ID of the patch that should go into the slot. The user can print the design on real paper, cut out the patches and slots, and assemble the sliceform according to the labels. An example result from our system is shown in Fig. 1.

Compared to the other variants of paper pop-ups, such as v-style and origamic architecture pop-ups, paper sliceform design poses a greater challenge in ensuring physical realizability, since real paper patches cannot intersect each other and there is the possibility of interlocking cycles that prevent assembility.

Contributions Our objective is to automate the design of realizable paper sliceforms. In achieving this, we have made the following contributions:

- We present geometric formulations for feasibility and physical realization of sliceform designs, which guarantee the stability, foldability as well as the local pairwise assembility of the resulting sliceforms.
- We describe a set of sufficient conditions that can be used to construct stable sliceform designs.
- We have developed an efficient algorithm to construct stable and flat-foldable sliceforms that approximate the given 3D solid models.
- We demonstrate a computational method to produce physically realizable sliceform designs. The algorithm guarantees local and pairwise assembility between any two patches, which is empirically shown to be equivalent to the global assembility.
- We present a novel 3D shape abstraction/simplification method using a generalized cylinder approximation and Reeb graph edge-merging.

2 Related Work

Recent studies in computational paper pop-ups have been inspired by phenomenal paper pop-up books created by paper engineers [3], [4]. While other forms of papercraft, such as origami [5], have been extensively studied in the mathematical and computational setting, studies specifically on paper pop-ups are scarce. Paper pop-ups however differ significantly from origami, since its construction allows cutting and the use of multiple sheets of paper that leads to more complicated structures. On the other hand, the types of folding in pop-ups are much more restricted; as such, formulations for origami cannot be easily ported.

Pop-up books use a variety of mechanisms. These mechanisms are usually simply categorized into those that consist of a single connected piece or multiple pieces of paper [6]. The book by Sharp [7] is among the few devoted specifically to sliceform pop-ups.

Computational Paper Pop-ups Early works in computational paper pop-ups mostly focus on explaining the geometric properties and on the simulation of specific pop-up mechanisms, such as v-style folding mechanism [8], [9], [10]. More recently, a general class of v-style pop-ups has been formulated by Li et al. [11]. Origamic architecture pop-ups have also been studied recently [12], [13]. In [8], [9], [11], [13], mathematical formulations have been made to allow computer aided design of pop-ups, in which users are offered a set of primitive structures to build virtual v-style or origamic architecture pop-ups. Feedback on the validity and simulation of folding/closing of the pop-up designs are often provided in real-time, such as in the interactive systems developed by Iizuka et al. [14] and Hendrix and Eisenberg [15]. Another interesting system has also been demonstrated by Hoiem et al. [16] that takes a photo and converts it into a simple paper pop-up.

Except for the work by Mitani and Suzuki [17], there has been very little research devoted to the study of paper sliceforms. Although sharing similar foldability problem with v-style and origamic architecture pop-ups, sliceforms require additional treatment for the physical realizability of the designs. Algorithms for automatic design of v-style and origamic architecture pop-ups have already been proposed [11], [12]. However, to the best of our knowledge, works in automatic design of paper sliceforms have not been reported. As shown in our later discussion, due to the property that they only contain interlocking cross-sections of the models, paper sliceforms are significantly more difficult to design automatically. Existing commercial CAD software tools for sliceform design [1], [2] are still far from automatic and are even unable to check for the validity of the designs.

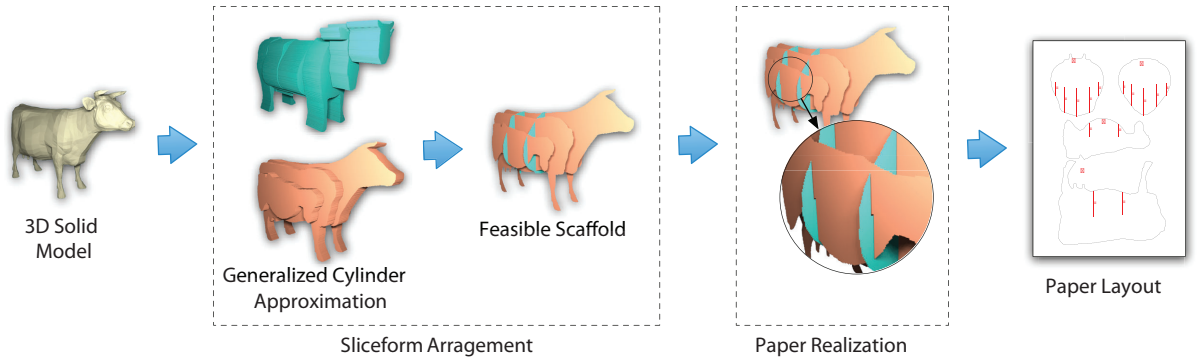

Fig. 4. The main steps in our algorithm.

Shape Simplification and Abstraction By reducing an input 3D model to a structure consisting of a small set of patches, sliceform pop-up design can be considered a form of model simplification or abstraction. Many existing works provide techniques to simplify a model by approximating its surface with simpler representations [18], [19]. A similar approach that can abstract human-made models has been proposed by Mehra et al. [20]. However, these approaches cannot be directly adapted to our problem as sliceforms are much more limited in depicting the surfaces of the models, unlike papercraft toys [21]. Simplification can also be done by fitting the model's volume with a small set of simple solid primitives [22]. This is closely related to our approach since we approximate the model using *generalized cylinders*. An interesting work has recently been reported by McCrae et al. that helps explain how human visual perception abstracts a 3D model to a set of planar cross-sections [23]. Although their approach is not used in our work, it may be useful in the future for enhancing our sliceform abstraction quality.

There have been studies on fabricating 3D abstractions into real physical models that use planar cutouts made out of wood, plastic or cardboard. The resulting models can be designed in a semi-automatic [24] or automatic manner [25]. Similar to sliceforms, such models are also made of intersecting planar patches. However, sliceforms require more complex stability conditions, since intersecting patches can rotate freely with respect to each other. Sliceforms also have to be flat-foldable. Specifically in [25], the simpler stability condition (i.e. the structure is stable as long as every patch intersects with at least one other patch) allows the algorithm to split patches without the need to re-evaluate and correct instability. The splitting of patches also allows their method to avoid the interlocking-ring configuration, which our method has to explicitly handle without splitting patches.

3 Overview and Formulations

Our method generates a paper sliceform design from a 3D solid model by performing the following steps:

1) **Sliceform Arrangement.** It finds the 2D shape and arrangement of the patches as well as their intersections such that the resulting paper sliceform is a good approximation of the input solid model, is stable and can be completely folded flat by moving any two non-coplanar patches.
2) **Paper Realization.** It determines how slots (or slits) along the intersections of the patches should be cut out so that they can be physically assembled to the target arrangement.

The overview of our algorithm is shown in Fig. 4. In this section, we describe the geometric conditions for a valid paper sliceform design. The algorithm details of each step are presented in Sections 4 and 5.

Inspired by the work of [11], we formulate a sliceform, or a patch arrangement, as a scaffold.

Definition 1. *A* scaffold *is a set of planar polygonal patches in 3D. These patches may have holes and intersect each other. Each line segment in the intersection of two patches is a* hinge.

If all the patches of a scaffold are parallel to only two directions, we call it a *sliceform scaffold*.

3.1 Feasible Scaffold

A proper patch arrangement must satisfy two important properties—it can be folded flat and the flattening does not require any extra forces besides moving any two of its non-coplanar patches. We call such arrangement a *feasible scaffold*. Assuming paper is rigid and has zero thickness, let the scaffold domain be $\mathbb{S}$, we define

Definition 2. *A* folding motion *from a scaffold S to another scaffold S' is a continuous mapping $g : [0, 1] \to \mathbb{S}$ such that*

- *$g(0) = S$ and $g(1) = S'$.*
- *g maintains the rigidity of the patches of S, for any $t \in [0, 1]$.*
- *g maintains the positions of the hinges on the patches of S, for any $t \in [0, 1]$.*

If such a folding motion exists, S' is said to be foldable *from S.*

Definition 3. *A scaffold S is said to be* stable *if*

- *At least two patches of S intersect each other.*
- *For every two non-coplanar patches p_1, $p_2 \in S$, there is no other scaffold $S' \neq S$ foldable from S while p_1, p_2 are kept stationary.*

A feasible scaffold is one that is both stable and foldable to a "flat" scaffold. Formally,

Definition 4. *A scaffold S is said to be* feasible *if*

- *There exists a scaffold S' foldable from S, such that the acute angle θ between every two intersecting patches of S' satisfies $0 < \theta < \epsilon$, with ϵ arbitrarily small. The respective folding motion is called* flat-folding *or* flattening *and S is said to be* flat-foldable.
- *For every $t \in [0, 1]$, the intermediate scaffold $g(t)$ during the deformation of S to S' is stable.*

Technically, our feasible scaffold formulation allows a wide set of patch arrangements. However, in this work, we are interested in the class of sliceform scaffolds, for which we are able to prove that

Proposition 1. *If a sliceform scaffold is stable, it is feasible.*

Proof: First, we show that a stable sliceform scaffold S is flat-foldable. Let p_1, p_2 be two arbitrary intersecting patches of S, and the angle between them be θ. We consider a cross-section of S on a plane $\mathcal{L}$ perpendicular to both p_1 and p_2. On $\mathcal{L}$, each hinge of S is projected to a point. Let O be the intersection between $\mathcal{L}$, p_1 and p_2. We choose $\vec{e}_1$, $\vec{e}_2$ as two unit vectors on $\mathcal{L}$ parallel to p_1 and p_2 respectively.

As the scaffold is a sliceform, every patch is parallel to p_1 or p_2. Furthermore, to satisfy the scaffold stability, no patch is"floating" or "dangling". In other words, each patch must intersect another and form an acute angle θ. By transforming all the angles from θ to ϵ, we are able to fold the scaffold flat. It can also be easily shown that no patch is obstructed during this folding motion. We represent each point of S in the $(O, \vec{e}_1, \vec{e}_2)$ coordinate system as $(t_1, t_2) = t_1\vec{e}_1 + t_2\vec{e}_2$. As all the patches remain parallel to p_1 and p_2 while they are folded, the coordinate of every point remains unchanged, and hence no collision occurs.

Besides, during the folding motion, each intermediate scaffold S' is stable. We prove it by contradiction. Assume there are two different scaffolds at angle θ' that share the same set of patches and hinge positions along each patch. There exists a hinge that can be represented by two coordinates $(t_1, t_2) \neq (t'_1, t'_2)$. By transforming the acute angles from θ' back to θ, the hinge can also be represented by both (t_1, t_2) and (t'_1, t'_2) in S, which violates its stability. □

As a result, it suffices to find a stable sliceform scaffold to guarantee a feasible patch arrangement.

3.2 Scaffold Realization

In practice, it is physically impossible to have paper patches intersecting each other as in a scaffold, hence, slots or slits are cut on these patches along their intersections such that the patches can be slotted together to physically realize the intersections. Let a *slot* be a straight-line cut having width $\epsilon > 0$, we define

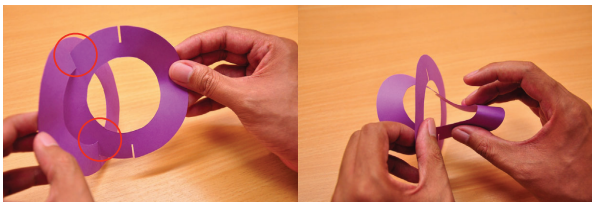

Fig. 5. A patch is bent to go through a hole of another patch.

Definition 5. *A* realization *of a scaffold S is another scaffold S' that satisfies*

- *S and S' have the same number of patches.*
- *Every patch $p'_i \in S'$ is a sub-patch of patch $p_i \in S$, formed by cutting out slots along the hinges of p_i.*
- *All the patches of S' do not intersect each other.*

The first condition maintains that each patch of S is not split by the cuts since we do not discard any patches from S'. The third condition prevents the realized patches from colliding once they are in position.

Besides avoiding collision between any two patches in a realization, we must also ensure that their *assembly process* exists. Intuitively, we consider an assembly process as a collision-free motion in which two patches, originally placed very far away from each other, are brought towards their target states.

Up to this point, we still inherit the assumption that paper has zero thickness. However, its rigidity may forbid assembility. We therefore assume that paper can be bent during assembly. This assumption is common in computational origami [5]. By bending a patch, for example, we are able to squeeze it through a hole of another patch during assembly. The process is shown in Fig. 5.

Formally, we define

Definition 6. *An* assembly motion *of patches p_1 and p_2 of a realization S is a continuous deformation of p_1 and p_2, denoted $g(p_1, t)$ and $g(p_2, t)$, such that*

- *$g(p_1, t)$ (and $g(p_2, t)$) preserves the geodesic distance between any two points on the surface of p_1 (and p_2), for any $t \in [0, 1]$.*
- *$g(p_1, 1) = p_1$ and $g(p_2, 1) = p_2$.*
- *$||g(p_1, 0) - g(p_2, 0)||_{\min}$, the minimum Euclidean distance between any two points on p_1 and p_2 respectively, is greater than any arbitrarily large distance d.*
- *$g(p_1, t)$ does not intersect $g(p_2, t)$, for any $t \in [0, 1]$.*

Finally,

Definition 7. *A realization is said to be* valid *if there exists an assembly motion between any two of its patches.*

Note that, given a feasible scaffold, all its realizations are physically flat-foldable. From our observations, there always exist trivial assembly sequences in which the patches do not block one another. For example, we can assemble the patches according to their order in the slicing direction. Consequently, the existence of an assembly motion for every pair of patches is empirically proven to produce valid realization.

The above formulation defines a valid paper sliceform. To produce such a sliceform, our algorithm first computes a set of *generalized cylinders* to approximate the input model. Then it uses these cylinders to find a stable sliceform scaffold. By Proposition 1, this sliceform scaffold is automatically feasible. Then, slots are created along the hinges of the patches such that every pair of the resulting patches has an assembly motion. The resulting paper sliceform is guaranteed to have local pairwise assembility, which we believe is equivalent to the global assembility of the structure.

4 Sliceform Arrangement

In this work, we approximate the input model using a sliceform whose patches are perpendicular to each other. We call such an arrangement an *orthogonal sliceform scaffold*.

Finding an orthogonal sliceform scaffold that is feasible and closely resembles the input model is particularly challenging. Computing the smallest set of patches that satisfies the feasibility condition and meets a certain approximation quality threshold is inherently intractable. Even obvious non-optimal-solution approaches, such as incremental patch addition, incremental patch removal, and fixing, are computationally intractable. In the incremental addition approach, one or more patches are added to the scaffold in each step, always ensuring scaffold stability, and greedily minimizing the number of patches and maximizing the approximation quality. However, due to the potentially large number of combinations of patches to be considered for addition in each step, this approach is intractable. The incremental removal approach also has similar intractability. In the fixing approach, we start with a scaffold that meets the approximation requirement, yet may not be stable, and "repair" it by incrementally adding more supporting patches until its feasibility is satisfied. However, this is exactly the same intractability in the incremental approach.

Our algorithm is grounded on the idea of using a small set of *primitives* to represent the patches, such that the feasibility of a scaffold can be verified by considering only these primitives. This enables us to drastically reduce the search space for the feasible scaffold and makes the problem much more *tractable*.

More specifically, we first generate a dense set of patches that captures the model's geometric features.

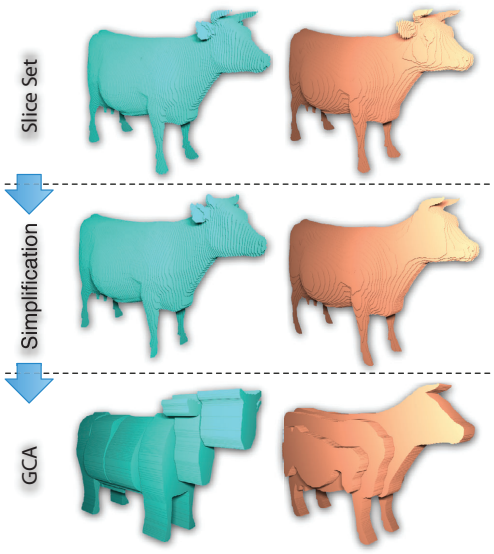

Fig. 6. Generalized cylinder approximation (GCA).

These patches are extracted from the model's cross-sections taken at small uniform-width interval along two orthogonal slicing directions chosen by the user. We call the collection of cross-sections in each slicing direction a *slice set*. Similar and consecutive patches in each slice set are grouped to form a *generalized cylinder* that approximates the model's volume occupied by the patches. Each of these cylinders takes the union of patches in the group as the shape of its cross-section, and the slicing direction as its axis. We call this step *generalized cylinder approximation*, as illustrated in Fig. 6.

Then, the target feasible scaffold is computed by using each cylinder's cross-section to create a set of patches within the cylinder, such that the patches from all the cylinders form a stable scaffold. Our algorithm can efficiently achieve this by checking for stability conditions only between parts of cylinders that overlap each other.

To facilitate our next discussion, let p and p' be two parallel patches, we denote $p \oplus p'$ as the patch created from the union of p and the projection of p' along its slicing axis direction onto the plane of p. Note that $p \oplus p' \neq p' \oplus p$. Moreover, if P is a set of patches p_i parallel to p, then $p \oplus P = \bigcup_{p_i \in P}(p \oplus p_i)$. We also denote

- $z(p)$ as the position of patch p along its slicing axis.
- $\mathrm{area}(p)$ as the area of patch p.
- Δs as the slicing interval to produce the slice sets.

4.1 Generalized Cylinder Approximation

4.1.1 Approximation Fundamentals

We consider the simple case of partitioning a set of parallel patches $P = \{p_1, \ldots, p_N\}$, in which $z(p_i) < z(p_{i+1})$. If a sub-set $P_{ij} = \{p_i, p_{i+1}, \ldots, p_j\}$ of P is put into the same group, we construct the corresponding

generalized cylinder by taking $u_i = p_i \oplus P_{ij}$ and $u_j = p_j \oplus P_{ij}$ as its end caps. The error of cylinder-approximating P_{ij} can be measured by the volume disparity between its cylinder and the model's part occupied by the group, which is specified by

$$\mathrm{err}(i,j) = \Delta s \sum_{k=i}^{j} |\,\mathrm{area}(p_k) - \mathrm{area}(u_i)|. \qquad (1)$$

Note that $err(i,j)$ does not only measure the volume difference, but also the shape difference between the generalized cylinder and the group of patches being approximated. This is because these patches are completely inside the cylinder.

We seek to find the minimal number of groups that partition P, such that the total approximation error does not exceed a certain threshold τ_A. Let $D_{n,i,j}$ be the minimum error by approximating patches p_i to p_j by n groups. The search can be efficiently done by solving the following dynamic programming problem:

$$D_{n,i,j} = \begin{cases} \mathrm{err}(i,j) & ,\text{for } n = 1, \\ \min(\mathrm{err}(i,k) + D_{n-1,k+1,j}) & ,\text{for } n > 1, \\ & \quad i \le k < j. \end{cases} \qquad (2)$$

The algorithm stops when $n > N$ or there is an $n = n_0$ such that $D_{n_0,1,N} < \tau_A$. We also keep track of the value of k where $D_{n,i,j}$ reaches its minimum and finally trace back in order to construct the respective optimal partition, as well as the cylinders.

4.1.2 Topology Simplification

Note that each slice set is also the model's level set with respect to its slicing direction. Hence the model's topology along each direction can be captured by computing the *Reeb graph* from the respective slice set [26]. Based on this observation, we first construct the Reeb graphs for both directions, then separately partition the patches on each edge of these graphs using the generalized cylinder approximation method described above.

However, due to the geometric details on the model, each Reeb graph can have intricate topology, where many edges may contain only a few small patches. Directly partitioning on this graph is not efficient due to the large number of cylinders. Hence, prior to partitioning, we simplify the topology using a few Reeb graph edge-merging operations. Our approach is similar to [27], but we use the relative volume difference between two neighboring parts to decide whether to absorb the smaller part into the larger part.

Let e and e' be two edges in a Reeb graph, which correspond to the sets of patches $P(e) = \{p_1, \ldots, p_n\}$ and $P(e') = \{p'_1, \ldots, p'_m\}$ respectively, where $z(p_i) < z(p_{i+1})$ and $z(p'_i) < z(p'_{i+1})$. We say that

- e' is *left-merged* to e if patch p_1 is replaced by $p_M = p_1 \oplus P(e')$ and e' is removed from the graph.

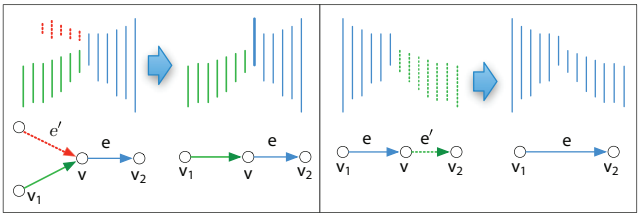

Fig. 7. Reeb graph edge-merging operations, where e' is merged to e.

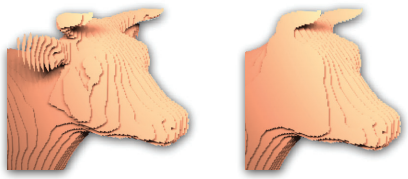

Fig. 8. (Left) Original slice set. (Right) Slice set after topology simplification.

- e' is *right-merged* to e if patch p_n is replaced by $p_M = p_n \oplus P(e')$ and e' is removed from the graph.

Let the weight $w(e)$ of each edge e be the sum of areas of all its patches and τ_M be a predefined threshold. We define the following edge-merging operations:

- If v is a vertex having *only one* outgoing edge (incoming edge) e and *more than one* incoming edge (outgoing edge), then any incoming edge (outgoing edge) e' of v having $w(e')/w(e) < \tau_M$ is left-merged (right-merge) to e.
- If v is a vertex having *only one* incoming edge e and *only one* outgoing edge e', then e and e' are merged by removing e' and replacing $p(e)$ by $p_M(e) = \{p_1, \ldots, p_n, p'_1, \ldots, p'_m\}$.

We illustrate the edge-merging operations in Fig. 7. Note that the case where v has multiple incoming and multiple outgoing edges cannot happen unless the input solid model is almost degenerate or the slicing interval is not small enough.

The topology simplification stops when no more edges can be merged. The objective of the simplification is to absorb very small parts into larger parts, independently in each slicing direction, so that the small parts are removed but their silhouettes (when viewed in the slicing direction) can still be preserved on the larger parts. This effect can be seen in Fig. 8, where the horns' silhouette is preserved in the slicing direction.

4.2 Finding Feasible Scaffold

4.2.1 Fundamentals

The input model is now approximated as a set of generalized cylinders. The orthogonal sliceform scaffold is to be made of patches that are cross-sections of these cylinders.

Let the span a cylinder c covers (on its axis) be denoted by $\mathrm{span}(c)$. We define

Definition 8. *A* frame *f of a cylinder c is a tuple (s_f, n_f), in which s_f is a sub-span in $\mathrm{span}(c)$, and $n_f \in \mathbb{N}^+$ is the minimum number of patches required in the span s_f.*

A cylinder may have zero or more frames, indicating where and how many of its cross-sections are used in a scaffold. If there are at least n_f patches taken from c in the span of a frame f, we say that f is *satisfied*. Formally,

Definition 9. *A set of frames F is said to be satisfied by a scaffold S, or S satisfies F, if*

- *Every patch of S is taken from at least one frame of F.*
- *Every frame of F is satisfied.*

Definition 10. *A frame set F is said to be stable if every scaffold S satisfying F is stable.*

A frame set actually represents a class of possibly-many scaffolds. If the frame set is stable, Proposition 1 says that the scaffold class it represents is indeed feasible. This is an important result since it enables us to indirectly find a feasible scaffold by constructing a stable frame set. In the followings, we describe a sufficient condition that makes a frame set stable. Our algorithm uses this condition to compute a stable frame set.

Definition 11. *Two frames f and f' of different slicing directions are said to* intersect *one another if every cross-section within f* intersects *every cross-section within f'.*

Claim 1. *Let $F = \{f_1, f_2, f_3, f_4\}$ be a set of frames, in which no cross-section within a frame is coplanar to any cross-section within any other frames. If f_1 and f_2 intersect both f_3 and f_4, then F is stable.*

Claim 2. *Let $F = \{f_1, f_2, f_3\}$ be a set of frames, where $f_3 = (s_3, n_3)$, and in which no cross-section within a frame is coplanar to any cross-section within any other frames. If f_1 and f_2 intersect f_3 and $n_3 \geq 2$, then F is stable.*

The proof of the above two claims is trivial, as the patches derived from F form a parallelogram and holding any two patches stationary will keep the rest from moving.

Definition 12. *Let F be a set of frames. A frame $f_1 \in F$ is said to be* semi-stable *with respect to another frame $f_2 \in F$ if either*

- *there exists a stable frame set $F' \subset F$ that contains both f_1 and f_2, or*
- *there exists a frame $f_3 \in F$ such that f_1 is semi-stable with respect to f_3 in F and f_3 is semi-stable with respect to f_2 in F.*

Definition 13. *Let F be a set of frames. $F' \subset F$ is said to be* semi-stable *in F if all its frames are semi-stable with respect to one another in F.*

Proposition 2. *Let $F = F_U \cup F_V$ be a frame set in which F_U and F_V are the sets of frames from the two slicing directions respectively. If F_U and F_V are semi-stable in F, then F is stable.*

Proof: When F_U is semi-stable in F, all the patches satisfying F_U may undergo different translational moves but must always remain parallel to each other. The same condition applies to F_V when it is semi-stable in F. As F_U and F_V are both semi-stable in F, every patch must intersect at least two patches from the other slicing direction. In this case, if we keep two non-coplanar patches p_1 and p_2 in F_U (or F_V) stationary, no patches in F_V (or F_U) can move, otherwise p_1 or p_2 has to move too. Similarly, if we keep one patch from F_U and one from F_V stationary, then all the patches from F are also stationary. Hence, the entire scaffold remains rigid if two arbitrary non-coplanar patches are held stationary. Thus, it is stable. □

Note that Proposition 2 allows the user to hold any two non-coplanar patches to keep the structure stable. This is different from Proposition 2 of [12], where the user must hold the two outer patches.

4.2.2 Algorithm Overview

The problem to produce a stable frame set is made easier by Proposition 2, where now we can just separately make F_U and F_V semi-stable in F. Our algorithm is also guided by the following common schemes that we observe in the making of sliceforms:

- **Construction for a dominant direction**. For a model whose distinctive features primarily appear in one particular direction, e.g. side-view of an animal model, the patches from this direction often are better at conveying the characteristics of the model, while the patches on the other direction are only used to support them.
- **Construction for both directions**. On the other hand, for a model whose characteristic features cannot be primarily viewed from a single direction, e.g. architectural models, patches from both directions are equally important for rendering and supporting the pop-up.

We let the user choose the desired scheme. For the following description, we denote the set of cylinders as $C = C_U \cup C_V$, where C_U and C_V are the cylinder sets from the two slicing directions U and V respectively. Accordingly, we denote the set of frames in C as $F = F_U \cup F_V$.

We call the *cylinder stabilizing operation* on two cylinders $c_1, c_2 \in C$ as the search for a frame set $F = \{f_1, f_2, f_3, f_4\}$ or $F = \{f_1, f_2, f_3\}$ of C, satisfying either Claim 1 or 2, such that f_1, f_2 are from c_1, c_2 respectively. This operation is denoted by $F \leftarrow \textsc{Stabilize}(C, c_1, c_2)$.

4.2.3 Construction For a Dominant Direction

Let us assume that the dominant direction is U. Our algorithm can be summarized in two steps. In the first

step, we find a set of frames $F = F_U \cup F_V$ such that all the cylinders C_U have *at least one* frame in F_U, and F_U is semi-stable. In the second step, we make F_V semi-stable. We illustrate the algorithm in Fig. 9 and discuss the two steps in details below.

Algorithm 1: F_U STABILIZATION

Data: Cylinder set C.
Result: A frame set F, in which F_U is semi-stable.

$F \leftarrow \emptyset$; $P \leftarrow \{c_0\}, c_0 \in C_U$;
while $P \neq C_U$ **do**
  find $c_1 \in P$ and $c_2 \in C_U \setminus P$, such that $F' \leftarrow$STABILIZE(C, c_1, c_2) is not empty;
  $P \leftarrow P \cup \{c_2\}$; $F \leftarrow F \cup F'$;
end
foreach $c \in P$, *c has more than one frame* **do**
  sort the frames of c by their spans;
  foreach *pair of consecutive frames* f_i, f_j *of* c **do**
    find an α-support H for f_i, f_j;
    $F \leftarrow F \cup H$;
  end
end

The pseudocode for the first step of our algorithm is given in Alg. 1. It iteratively adds frames of C_U cylinders into F, until all of them have at least one frame in F.

Stabilizing using α-support. If a cylinder $c \in C_U$ is chosen more than once in the previous stabilization step, it has more than one frame. This set of frames has to be stabilized in order to make F_U semi-stable, which is done by building an α-support for each pair of consecutive frames. For two consecutive frames f_i and f_j of c, an α-support between them is a frame set $H = \{h_1, h_2, \ldots, h_{2n+1}\}$, where

- $h_1 = f_i$ and $h_{2n+1} = f_j$.
- h_{2k} is a frame of a cylinder in C_V, such that h_{2k} requires at least 2 patches.
- h_{2k+1} is a frame of c.
- The frame set $\{h_{2k-1}, h_{2k}, h_{2k+1}\}$ is stable for all $k \leq n$.

Additional new frames may need to be added in cylinder c, between f_i and f_j, to construct an α-support. Clearly, if all frames of c have an α-support between them, they are semi-stable with respect to each other.

Note that an α-support always exists between f_i and f_j. If there is no cylinder in C_V that intersects both f_i and f_j, it is always possible to find a chain of adjacent cylinders in C_V where they all intersect c, the first one intersects f_i and the last one intersects f_j. In this case, a new frame is created in cylinder c to "connect" and support the two frames from each pair of consecutive cylinders in the chain.

In the second step of the scaffold construction, we make frame set F_V semi-stable. However, if α-supports are used, additional frames will be added to F_U, which then have to be made semi-stable again. Fortunately, each frame of F_U up to this point must intersect at least two frames of F_V and vice versa. Therefore, making F_V semi-stable can be achieved simply by using the condition in Claim 2, where it is sufficient to increase the minimum number of required patches of a few frames in F_U to 2. The algorithm of the second step is described in Alg. 2.

Algorithm 2: F_V STABILIZATION

Data: A frame set F, where F_U is semi-stable.
Result: F, having both F_U and F_V semi-stable.

$Q \leftarrow \{f_0\}, f_0 \in F_V$;
while $Q \neq F_V$ **do**
  find $f_1 \in Q$ and $f_2 \in F_V \setminus Q$, such that f_1 is semi-stable with respect to f_2 in F;
  if *found* **then**
    $Q \leftarrow Q \cup \{f_2\}$;
  else
    foreach $f_3 \in F_U$ **do**
      if f_3 *intersects* $f_1 \in Q$ *and* $f_2 \notin Q$ **then**
        $n_{f_3} \leftarrow 2$; $Q \leftarrow Q \cup \{f_2\}$;
      end
    end
  end
end

Note that in Alg. 1, before the computation of α-supports, it is always possible to have every cylinder in C_U contribute at least a frame to F. This is because for any two adjacent cylinders in C_U, there always exists another cylinder from the other direction that intersects both cylinders, and thus can provide a frame that stabilizes the frames of the two cylinders.

4.2.4 Construction For Both Directions

The algorithm is similar to that of the previous scheme. However, we require all the cylinders to have at least one frame in F after the first step. Again, it is important that we only use α-supports to stabilize the frames of one direction, while those of the other direction are stabilized by the method in Alg. 2.

4.2.5 Generating Scaffold

We finally generate the feasible scaffold S from F. For each frame f, we create n_f patches from the respective cylinder of f, and uniformly place them in s_f.

Each cylinder may have multiple frames whose spans may overlap. Separately generating patches from each of these frames may produce unnecessarily many patches. We avoid it by applying a simple frame-reduction scheme prior to computing the scaffold. If f_1 and f_2 are two overlapping frames in the same cylinder, we substitute them by f_3, where $s_{f_3} = (s_{f_1} \cap s_{f_2})$ and $n_{f_3} = \max(n_{f_1}, n_{f_2})$.

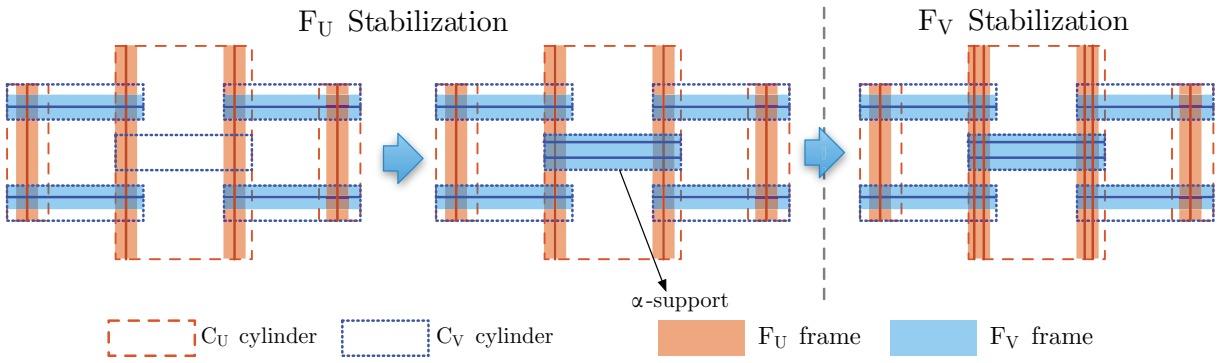

Fig. 9. (Left) Cylinders in C_U are ensured to have at least one frame. (Middle) α-supports are used to make F_U semi-stable. (Right) F_V is made semi-stable by modifying some frames of F_U.

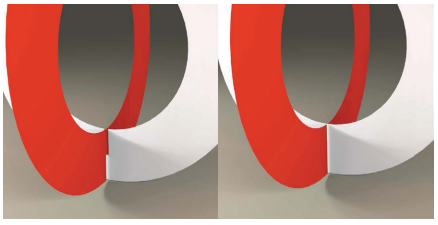

Fig. 10. Four types of slots. (Left) White: **up**, orange: **down**. (Right) Orange: **cut-through**, white: **none**.

5 Paper Realization

Given two patches p_1 and p_2, their intersection may contain multiple segments, each of which corresponds to a hinge. At each hinge, one slot must be cut out from each respective patch in order for them to be assembled at the hinge. We define four types of slot as illustrated in Fig. 10: **up**, **down**, **cut-through** and **none**. If a slot is **up** or **down**, it is cut upwards or downwards respectively along the hinge's direction, starting at the mid-point of the hinge and ending at the patch's boundary or the mid-point of the next hinge on the patch, whichever comes first. If a slot is a **cut-through**, the hinge on the patch is fully cut. If the type is **none**, the patch remains intact at the hinge. At a hinge of two patches p_1 and p_2, the only possible combinations of their slots are: (**up**, **down**), (**down**, **up**), (**cut-through**, **none**) and (**none**, **cut-through**).

By making slots this way, the patches will not collide with each other along their hinges. However, collision-free hinges still do not ensure assembility of the patches, since depending on the slot combination, two patches may interlock each other and prevent them from being assembled, similar to the result of Autodesk 123D Make in Fig. 3. We show that

Proposition 3. *There exists an assembly motion g for two patches p_1 and p_2 if no cycle in the Reeb graph of p_1 interlocks any cycle in the Reeb graph of p_2.*

Proof: We prove that given two patches that have no interlocking cycles at the target sliceform state, one of them can be detached from the other without collision, hence the reversed motion is an assembly motion.

Let a and b be two cycles in patches A and B respectively. Let H be the intersection line between the two planes that contain A and B. If a does not interlock b, it must satisfy the condition that the number of intersections of a and H that are inside and outside b must both be even. Furthermore, we say that a is *completely inside* (or *completely outside*) b if all its intersections with H are all inside (or outside) of b.

We call a group of cycles a *dependent group* if each of its cycles shares a common edge with at least one other cycle in the group. A Reeb graph can be partitioned into disjoint dependent groups.

Consider two patches A and B, in which every cycle has at most two intersections with H. It can be observed that if all the cycles of A and B do not interlock, any dependent group s of A must be either completely inside or outside any cycle $b \in B$ and vice versa. Otherwise, we can always find a cycle $a \in s$ that has exactly two intersections with H, of which one is inside and one is outside b (odd number of intersections), which does not satisfy the non-interlocking condition.

With the assumption that paper has zero thickness, such a dependent group $s \in A$ can be packed to as small as needed by folding it locally, to avoid collision with B. Clearly, this folding is independent of other groups $s' \in A$, and is also independent of B. Therefore, A can be "untangled" from B by packing every dependent group in A.

Given non-interlocking cycles in patches A and B, each can be made to intersect with H at at most two points by locally folding the patch as illustrated in Fig. 11. Therefore, by combining local folding motion and dependent group packing motion, it is possible to "untangle" the two patches. Hence, the reversed motion is the assembly motion. □

A valid realization of two patches p_1, p_2 is therefore a combination of their slots such that p_1 and p_2 do not contain any interlocking pair of cycles. It is obvious that checking all pairs of cycles on the Reeb graph of p_1 and p_2 for this condition is impractical. Fortunately, we need to consider only those containing a patch

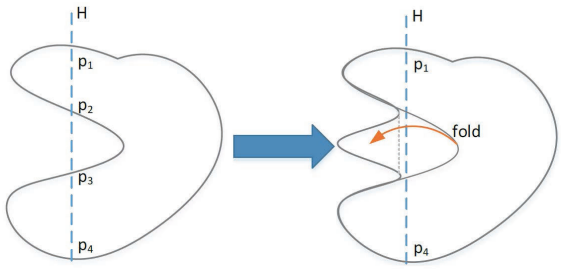

Fig. 11. Local patch folding.

hole and also intersecting one of the hinges, since they are the cycles that can potentially interlock with one from the other patch. Let G_1 and G_2 respectively be the Reeb graphs of p_1 and p_2. It can be seen that the cycles in G_1 and G_2 that intersect the hinges necessarily represent all the holes of interest. We say that a slot combination satisfies a pair of cycles if the cycles do not interlock with respect to the combination.

Any cycle in a graph can be constructed from a small set of cycles which forms the graph's cycle basis [28]. Furthermore, if every cycle in a cycle basis of a Reeb graph does not interlock any cycle in a cycle basis of another Reeb graph, all their derived cycles do not interlock either. Consequently, a valid slot combination between two patches is the combination that satisfies all cycle pairs from their cycle bases.

In addition, a combination that satisfies a cycle pair may not satisfy the other. We call this situation a *conflict*. If there is no slot combination that can resolve the conflict, we have to break a cycle of the cycle basis of one of the patches by a **cut-through** slot.

Our algorithm works by rejecting conflict combinations as illustrated in Fig. 12. Given two patches p_1, p_2, we construct their Reeb graphs G_1 and G_2 and a set T of the satisfactory slot combinations. For each cycle pair from the cycle bases of G_1 and G_2, we reject the combinations in T that conflict with the current pair. If T is empty after this step, which means there was an unresolvable conflict, we break one of these cycles, update the respective patch's cycle basis and restart the algorithm. On the other hand, if T is not empty, we add to T the new combinations that satisfy the pair. The algorithm stops when all pairs are satisfied, and one of the combinations in T is chosen as the slot combination. Finally, the patches and their slots are laid out on a plane for printing. We summarize our realization algorithm in Alg. 3.

In practice, for two intersecting patches, the number of common hinges are rarely more than a few and the cycle bases are usually small. Furthermore, once a slot is determined as **none** or **cut-through**, it does not contribute to the interlocking of the patch anymore, and we need to consider only the **up** and **down** slots for the slot combinations. Therefore, for a pair of cycles, the number of tested combinations is also restricted to a few. Additionally, the more hinges and cycles in the patches' cycle bases, the higher the chance conflicts

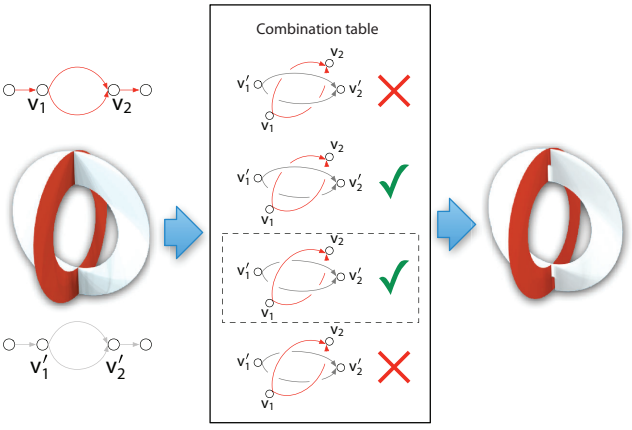

Fig. 12. (Left) Two patches and their Reeb graphs. (Middle) Slot combinations are checked and rejected from combination table. (Right) A valid combination is chosen as the realization of the two patches.

Algorithm 3: REALIZATION

Data: Patches p_1 and p_2.
Result: Valid slot combination.

$G_1 \leftarrow$REEBGRAPH(p_1), $G_2 \leftarrow$REEBGRAPH(p_2);
$T \leftarrow \emptyset$;
repeat
  $resolvable \leftarrow$**true**; $first\_pair \leftarrow$**true**;
  foreach *cycle pair* (b_1, b_2) *of* G_1, G_2*'s cycle bases* **do**
    $T' \leftarrow$ all slot combinations of (b_1, b_2) that do not interlock;
    if $first\_pair$ = **true then**
      $T \leftarrow T'$; $first\_pair \leftarrow$**false**;
    end
    reject combinations in T conflicting with T';
    if $T = \emptyset$ **then**
      break b_1 and update G_1;
      $resolvable \leftarrow$**false**;
      break;
    else
      add compatible combinations in T' to T;
    end
  end
until $resolvable$;

occur, which eventually reduces the number of cycles in the cycle bases. Therefore, the algorithm, though theoretically having exponential time complexity with respect to the number of hinges, is practical.

Note that even though we use **cut-through** slots, no patch will be split (into two or more subpatches). This is because a **cut-through** slot is normally applied to only one hinge of a cycle where conflict happens. Once cut through, the cycle c is no longer a cycle and will not be cut again. However, if it becomes part of a larger cycle, it is still fine to cut c again because the larger cycle will still connect the two pieces of c together. The same reasoning applies to the large

cycle and so on. Also note that the generation of slots does not affect the stability between any two intersecting patches, since they always have at least one (**up**, **down**) slot combination connecting them.

6 Implementation

Our automatic sliceform design application takes a water-tight 3D model in Wavefront OBJ format as input and produces a 2D layout of the generated sliceform in Adobe PDF format. The user first specifies the input 3D model as well as the slicing directions and thresholds (τ_A and τ_M). Afterwards, the system generates a feasible scaffold, its valid realization and finally a 2D layout that can be printed, cut and assembled into a physical sliceform. The layout shows the outline of the patches as well as where to cut the slots on each patch. Every patch and slot is labeled accordingly for easy construction.

To obtain the cross-sectional slices of the input 3D model, we use a method similar to the shadow volume technique. At each slicing position, we use the in-out parity of an image ray parallel to the slicing direction to determine if a point on the slicing plane is inside or outside the model. We then use connected-component labeling to identify contiguous regions that become the model's patches.

The slices are represented as 2D billboards for both the visualization and computation. The image on each billboard is a binary image, where the non-zero pixels represent the regions that belong to the patches. The union of a set of patches can be computed simply by taking the union of their binary images.

In the generalized cylinder approximation, naive implementation of the dynamic programming problem in (2) will have $O(M^2N^2)$ time complexity for each iteration, where M is the image size ($M \times M$) and N is the number of slices in the slicing direction. This complexity is dominated by the computation of the union of the patch images. We also observe that the union of a set of $2k$ images $p = \cup_{i=1}^{2k} p_i$ is also equivalent to $p = \cup_{i=1}^{k} p'_i$, where $p'_i = p_{2i-1} \cup p_{2i}$, therefore a binary tree data structure is used to solve (2), and it reduces the time complexity to $O(M^2 \log N)$.

After generating a realizable scaffold, the system produces a 2D design layout by printing the binary images of the patches (with their slots and labels) on a page in a left-to-right and top-to-bottom manner. Multiple pages of layout may be produced.

7 Results

Our experiments were run on a PC with Intel Core i5 CPU and 4GB of RAM. We set the image size to 512×512 for each slice and used 128 slices on each direction to generate the slice sets for the input model. The program used up to 500MB of memory and takes 4–7 minutes of running time for each model we tested. The most time-consuming computation was the generalized cylinder approximation, which accounts for up to 90% of the running time due to the large number of pixel operations.

Fig. 13. Cow sliceforms using different values of τ_A and number of dominant slicing directions: (left) 0.05 and 1, (middle) 0.05 and 2, and (right) 0.015 and 1.

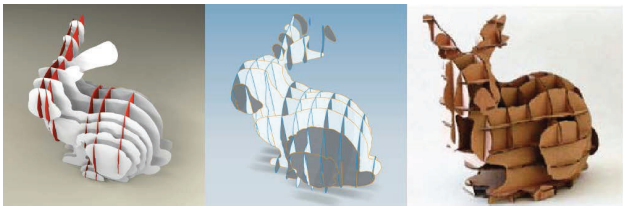

Fig. 14. Comparison of results generated by (left) our algorithm, (middle) Autodesk 123D Make, and (right) crdbrd [25].

The input models we used can be categorized into two groups: organic models (Bunny, Kitten, Cow, Bird, Dinosaur, Mother and Child Statue, and Armadillo), and architectural models (Capitol Building, Machine Part, Hollow Cube, Hollow Sphere and Torus). We chose one dominant direction for the organic models, and two dominant directions for the architectural ones. The topology simplification error threshold τ_M ranged from 0.02 to 0.05, while the approximation threshold τ_A was kept at 0.1.

In Fig. 13, we show the effect of the approximation threshold τ_A and the number of dominant directions. Our other results are shown in Fig. 1 and Fig. 18, where we chose the parameter values that produced the best sliceform for each input 3D model.

We have observed that all the sliceform scaffolds produced by our algorithm are feasible, and the overall shapes mostly have good resemblance to the input models. Using the Reeb graph edge-merging operations, our algorithm is able to disregard smaller geometric details in the more complex organic models while still retaining good level of abstractions when the sliceforms are viewed in the dominant slicing direction. For example, the silhouettes of the horns of the Cow, and the ears of the Bunny and Kitten are all preserved in the final sliceforms. We have also observed that our algorithm does not produce many unnecessary patches. In most parts of the models, the algorithm generates only those patches mandatory to maintain the stability of the structures.

Fig.14 compares the result from our algorithm with that of Autodesk 123D Make and crdbrd [25]. Here we observe that the sliceform generated by our algo-

rithm is stable unlike the output of Autodesk 123D Make. The result of crdbrd seems to have better approximation of the original 3D model, probably because it is not restricted to only two sets of parallel patches as flat-foldability is not required. Besides, our sliceform scaffolds have stability requirement that is more stringent than that of crdbrd.

We demonstrate the ability of our algorithm in guaranteeing assembility by making real paper sliceforms from the designs produced by our algorithm. These real sliceforms are shown in Fig. 15. It took 30 minutes (the Cow) to 3 hours (the Bunny) to cut and assemble each model. Other than the Cow model, all the models have complex patch topologies, e.g. the windows in the Capitol Building and the interior hole in the Hollow Sphere. Our algorithm avoids interlocking cycles on the patches in such cases.

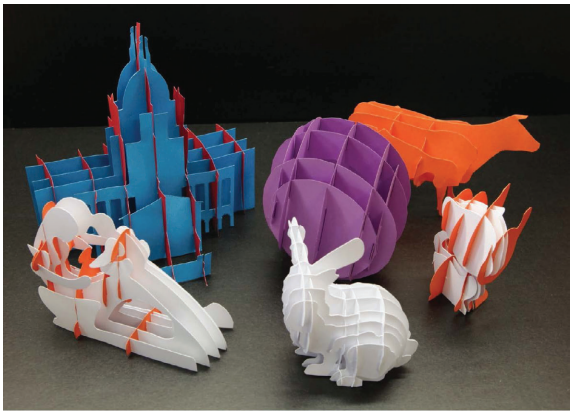

Fig. 15. Real paper sliceforms made from the designs produced by our algorithm.

However, we notice that for some models, e.g. the Capitol Building and the Mother and Child Statue, some patches are slotted very near to their boundaries as illustrated in Fig. 16(a). This results in weak points where patches can easily be torn off during assembly. These patches also form weak hinges that affect the overall strength of the sliceform structure. A better algorithm will need to constrain the minimum distance between slots and patches' boundaries during the scaffold generation. Another possible solution is to dilate the patches to increase the slot-to-boundary distance.

Since we only generate enough patches that satisfy the set of frames, long cylinders may contain only a few patches, which makes the resulting sliceform look sparse as shown in Fig. 16(b). More uniformly and densely distributed patches can be added by applying a global optimization in the scaffold generation step to find a valid scaffold that is constrained by the minimum and maximum distances between its adjacent parallel patches. Some input models may also be difficult to model with only two sets of parallel patches, as some important features of the models may not be oriented in the two dominant directions.

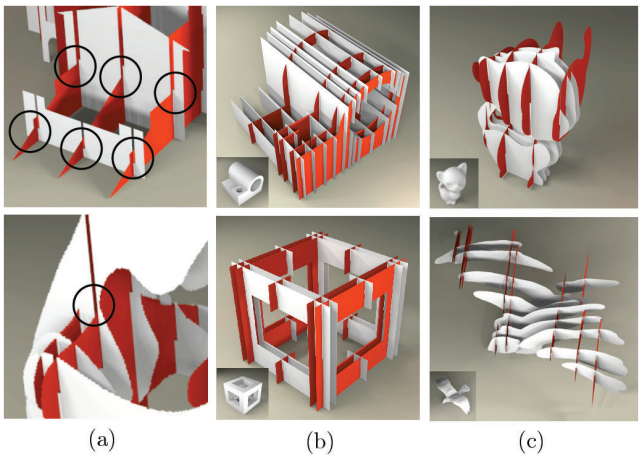

Fig. 16. (a) Slots are too close to patches' boundaries. (b) Patches are too sparsely distributed in long cylinders. (c) Unsuitable slicing directions.

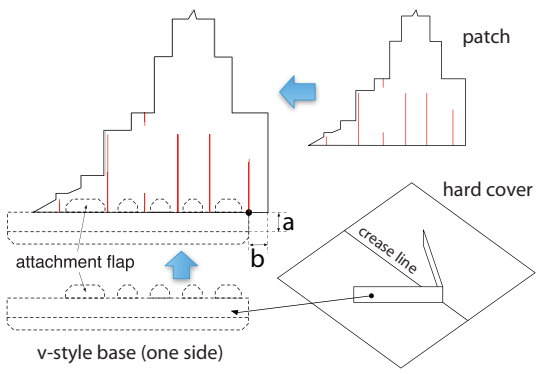

Fig. 17. Making a sliceform pop-up.

Such is the case of the head of the kitten model shown in Fig. 16(c), which is tilted slightly from its body and therefore no one patch can capture both ears of the head, which provide one of the most important visual cues for this model. The bird example in the same figure also demonstrates the undesired effect of using unsuitable slicing directions. For the bird model, however, a good sliceform can still be obtained with suitable slicing directions (see Fig. 18).

Pop-up Book Construction Sliceforms are also known as lattice-type folds, which is a popular folding mechanism employed in pop-up books. Fig. 17 illustrates how we can turn any paper sliceform into a paper pop-up, by attaching it to a hard cover using a v-fold. This v-fold has flaps that can be glued to two non-parallel patches of the sliceform and also two other big flaps that can be glued to the cover. The crease line of the v-fold is collinear with the intersection line of the two sliceform patches that it attaches to. When the hard cover is fully opened, the v-fold is opened at 90-degree angle. To avoid collision during folding, the height a of the v-fold is required to be at least the width b of the part not on the v-fold. This can be proven by considering a special case of the v-fold [11]. We have used this technique to make

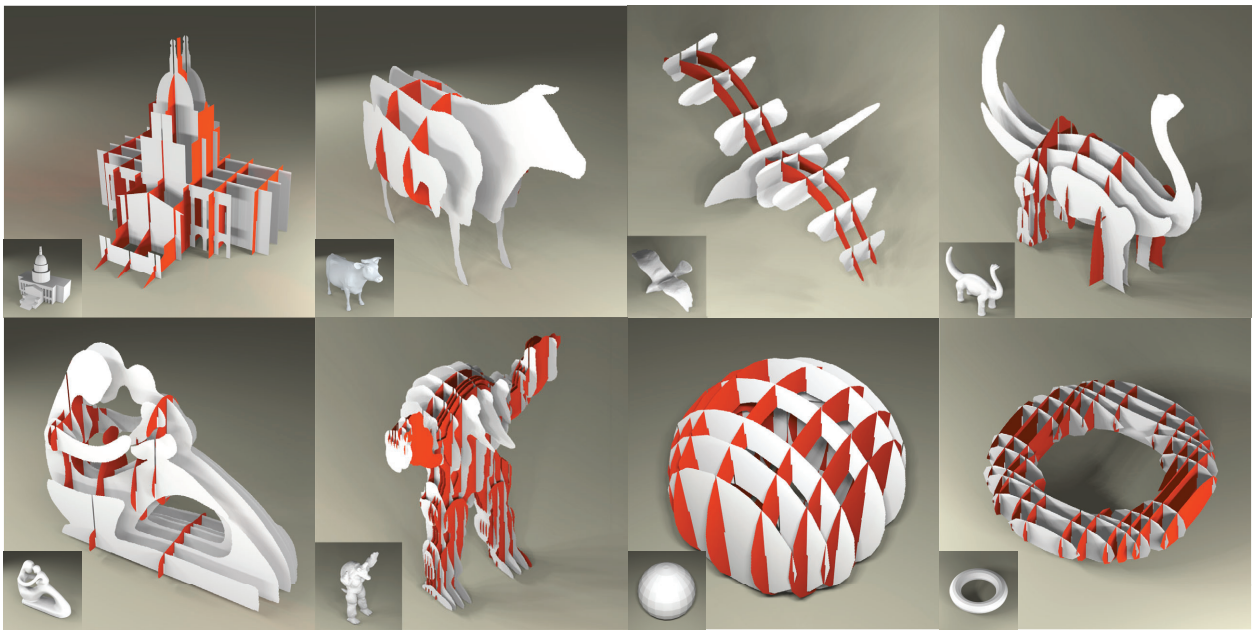

Fig. 18. Input models and sliceforms (from left to right, top to bottom): Capitol Building, Cow, Bird, Dinosaur, Mother and Child Statue, Armadillo, Hollow Sphere and Torus.

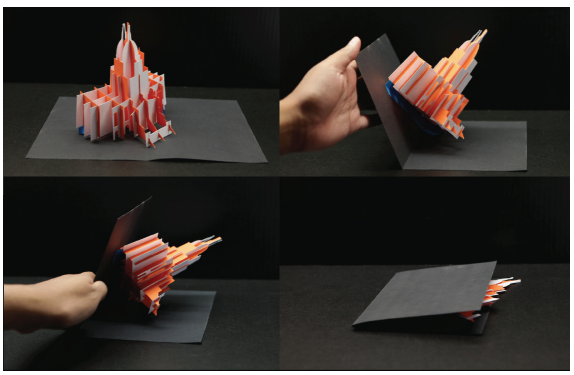

Fig. 19. A paper sliceform pop-up being flat-folded.

a paper sliceform pop-up of the Capitol Building. The pop-up and its ability to be folded flat is demonstrated by Fig. 19.

8 Conclusion and Discussion

We have presented a theoretical framework and a computational model for the automatic design of paper sliceforms from 3D solid models. In spite of the challenges in finding feasible scaffolds, we have made the problem tractable by the use of the generalized cylinder approximation and the frame set formulation. Another important notion in our approach is the avoidance of interlocking cycles in the paper realization problem. Experiment results have demonstrated the ability and robustness of our algorithm to generate feasible paper sliceform designs that can be physically assembled for a variety of input 3D solid models. Nevertheless, the global assembility of our sliceform designs has only been empirically validated, and formally proving it remains one of the outstanding challenges.

Our work may also open up interesting possibilities for future research in paper sliceforms. For example, given a class of feasible scaffolds, one may explore how to choose the best scaffold that maximizes the similarity between the sliceform and the input model. This will require the development of an effective and objective quality measurement metric. In addition, one can also investigate how the slots on a scaffold can be realized to ensure adequate strength, while still offering enough simplicity to the user to assemble the output sliceform.

Our algorithm constructs a scaffold using the base structures defined by Claim 1 and Claim 2. Although we have not encountered any models that cannot be approximated by these structures, some complex models may require other forms of scaffolds to best represent them. Therefore, from a theoretical perspective, it is interesting to consider extending these conditions in order to include larger and more complex structures to construct the sliceforms.

Our formulation assumes that paper can be freely bent during the assembly process. However, in practice, paper is bendable only to a certain degree, especially when part of a patch is already assembled and locked with other patches. This can make the assembly process difficult. Accurate modeling of this paper limitation will be helpful to producing designs that can be more easily assembled.

In addition, due to this assumption, the realization algorithm cannot be applied to thick or rigid material, such as wood or carton paper. Finding a realization method that can be used for these kinds of rigid material is another interesting direction for future research.

ACKNOWLEDGMENT

This work is supported by the Singapore MOE Academic Research Fund (Project No. T1-251RES1104).

REFERENCES

[1] Autodesk, "123D Make," 2012. [Online]. Available: http://www.123dapp.com/
[2] P. A. International, "SketchUp SliceModeler Plugin," 2009. [Online]. Available: http://www.public-art-international.com/catalog/product_info.php/products_id/200
[3] R. Sabuda and L. Carroll, *Alice's Adventures in Wonderland*, ser. New York Times Best Illustrated Books. Little Simon, 2003.
[4] M. Chatani, *Paper Magic: Pop-up Paper Craft : Origamic Architecture*, ser. The world of paper magic. Ondorisha Pub., 1988.
[5] E. D. Demaine and J. O'Rourke, *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. New York, NY, USA: Cambridge University Press, 2007.
[6] P. Jackson and P. Forrester, *The Pop-Up Book: Step-By-Step Instructions for Creating Over 100 Original Paper Projects*, ser. An Owl book. Henry Holt and Company, 1994.
[7] J. Sharp, *Surfaces: Explorations With Sliceforms*. Tarquin, QED Books, 2004.
[8] A. Glassner, "Interactive Pop-up Card Design. 1," *Computer Graphics and Applications, IEEE*, vol. 22, no. 1, pp. 79 –86, jan/feb 2002.
[9] ——, "Interactive Pop-up Card Design. 2," *Computer Graphics and Applications, IEEE*, vol. 22, no. 2, pp. 74 –85, mar/apr 2002.
[10] Y. T. Lee, S. B. Tor, and E. L. Soo, "Mathematical Modelling and Simulation of Pop-up Books," *Computers & Graphics*, vol. 20, no. 1, pp. 21 – 31, 1996, computer Graphics in Singapore.
[11] X.-Y. Li, T. Ju, Y. Gu, and S.-M. Hu, "A Geometric Study of V-style Pop-ups: Theories and Algorithms," in *ACM SIGGRAPH 2011*. New York, NY, USA: ACM, 2011, pp. 98:1–98:10.
[12] X.-Y. Li, C.-H. Shen, S.-S. Huang, T. Ju, and S.-M. Hu, "Popup: Automatic Paper Architectures from 3D Models," in *ACM SIGGRAPH 2010*. NY, USA: ACM, 2010, pp. 111:1–111:9.
[13] J. Mitani and H. Suzuki, "Computer aided Design for Origamic Architecture Models with Polygonal representation," in *Proc. of the Computer Graphics International*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 93–99.
[14] S. Iizuka, Y. Endo, J. Mitani, Y. Kanamori, and Y. Fukui, "An Interactive Design System for Pop-up cards with a Physical Simulation," *Vis. Comp.*, vol. 27, no. 6-8, pp. 605–612, Jun. 2011.
[15] S. Hendrix and M. Eisenberg, "Computer-assisted Pop-up Design for Children: Computationally Enriched paper Engineering," *Adv. Tech. Learn.*, vol. 3, no. 2, pp. 119–127, Apr. 2006.
[16] D. Hoiem, A. A. Efros, and M. Hebert, "Automatic Photo Pop-up," in *ACM SIGGRAPH 2005*. New York, NY, USA: ACM, 2005, pp. 577–584.
[17] J. Mitani and H. Suzuki, "Computer aided Design for 180-degree Flat Fold Origamic Architecture with Lattice-type cross sections." *J. Graphic Science of Japan*, vol. 37, no. 3, pp. 3–8, Sep 2003.
[18] D. Cohen-Steiner, P. Alliez, and M. Desbrun, "Variational Shape Approximation," in *ACM SIGGRAPH 2004*. New York, NY, USA: ACM, 2004, pp. 905–914.
[19] A. Sheffer, "Model Simplification for Meshing using Face Clustering," *Comp.-Aided Design*, vol. 33, no. 13, pp. 925–934, 2001.
[20] R. Mehra, Q. Zhou, J. Long, A. Sheffer, A. Gooch, and N. J. Mitra, "Abstraction of Man-made Shapes," in *ACM SIGGRAPH Asia 2009*. New York, NY, USA: ACM, 2009, pp. 137:1–137:10.
[21] J. Mitani and H. Suzuki, "Making Papercraft Toys from Meshes using Strip-based Approximate Unfolding," *ACM Trans. Graph.*, vol. 23, pp. 259–263, August 2004.
[22] J.-M. Lien and N. M. Amato, "Approximate Convex Decomposition of Polyhedra," in *Proc. of the 2007 ACM symposium on Solid and physical modeling*. New York, NY, USA: ACM, 2007, pp. 121–131.
[23] J. McCrae, K. Singh, and N. J. Mitra, "Slices: a Shape-proxy based on Planar Sections," in *Proc. of the 2011 SIGGRAPH Asia Conference*. NY, USA: ACM, 2011, pp. 168:1–168:12.
[24] Y. Schwartzburg and M. Pauly, "Design and optimization of orthogonally intersecting planar surfaces," *Computational Design Modelling*, pp. 191–199, 2012.
[25] K. Hildebrand, B. Bickel, and M. Alexa, "crdbrd: Shape fabrication by sliding planar slices," *Comp. Graph. Forum (Eurographics 2012)*, vol. 31, no. 2, pp. 583–592, May 2012.
[26] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci, "Loops in reeb graphs of 2-manifolds," *Discrete Comput. Geom.*, vol. 32, no. 2, pp. 231–244, Jul. 2004.
[27] H. Carr, J. Snoeyink, and M. van de Panne, "Simplifying flexible isosurfaces using local geometric measures," in *Proc. of the conference on Visualization '04*, ser. VIS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 497–504. [Online]. Available: http://dx.doi.org/10.1109/VISUAL.2004.96
[28] D. Hartvigsen and E. Zemel, "Is every cycle basis fundamental?" *Journal of Graph Theory*, vol. 13, no. 1, pp. 117–137, 1989.

Tuong-Vu Le-Nguyen received his B.Sc. degree from Vietnam National University in 2005. In 2012, he obtained his M.Sc. degree in Computer Science from the National University of Singapore. He has worked in a number of projects in the areas of computer graphics and computer vision. He is currently running a start-up company that provides cloud-based software solutions.

Kok-Lim Low is an Assistant Professor at the Department of Computer Science of the National University of Singapore (NUS). He holds a Ph.D. degree in Computer Science from the University of North Carolina at Chapel Hill, and received his M.Sc. and B.Sc. (Honours) degrees in Computer Science from NUS. His research interests include computational art, real-time rendering, and computational photography.

Conrado Ruiz Jr. is a Ph.D. candidate at the School of Computing of the National University of Singapore (NUS). He also obtained his M.Sc. degree from NUS and received his B.Sc. degree in Computer Science from De La Salle University (DLSU) - Manila. He is currently on leave from his faculty position at DLSU. He has co-authored papers in the areas of computer graphics and multimedia retrieval.

Sang N. Le is a Ph.D. candidate and a teaching staff at the School of Computing of the National University of Singapore (NUS). He obtained his B.Sc. in Computer Science, with a minor in Mathematics, from NUS in 2006. He has worked in a number of research areas, ranging from biomechanics and human motion synthesis to 3D reconstruction. His current focus is on automatic design of paper pop-ups.