MAKING SHAPES FOLDABLE

by

Zhonghua Xi
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Jyh-Ming Lien, Dissertation Director
_____	Dr. Mary Frecker, Committee Member
_____	Dr. Yotam Gingold, Committee Member
_____	Dr. Amarda Shehu, Committee Member
_____	Dr. Qi Wei, Committee Member
_____	Dr. Sanjeev Setia, Department Chair
_____	Dr. Kenneth S. Ball, Dean, The Volgenau School of Information Technology and Engineering

Date: _____ Spring Semester 2017
George Mason University
Fairfax, VA

Making Shapes Foldable

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Zhonghua Xi
Master of Science
George Mason University, 2015
Bachelor of Engineering
Shanghai Jiao Tong University, 2009

Director: Dr. Jyh-Ming Lien, Professor
Department of Department of Computer Science

Spring Semester 2017
George Mason University
Fairfax, VA

Copyright © 2017 by Zhonghua Xi
All Rights Reserved

Acknowledgments

First, I would like to thank my committee members: Jyh-Ming Lien, Mary Frecker, Yotam Gingold, Amarda Shehu and Qi Wei. In particular, I deeply appreciate Jyh-Ming Lien for serving as my advisor throughout the Ph.D. program and bringing this interesting and challenging research topic to me. I would also like to thank my co-authors: In-Suk Choi, Yue Hao, Yun-hyeong Kim (and her mother), Young J. Kim, Guilin Liu and Yanyan Lu for their hard work that made those publications possible. Thanks to several grad students from MASC research group for their advice and support in this process: Evan Behar, Yue Hao, Guilin Liu, Yanyan Lu, Arsalan Mousavian and Christopher Vo. Thanks to Huangxin Wang for for editing and support. Also, thanks to the wonderful staff at the George Mason University Computer Science Department for their help. Finally, I would like to thank my mom and dad for their love, sacrifice, encouragement and support that I can dedicate my time into research.

The author gratefully acknowledges the support of the National Science Foundation grant EFRI 1240459.

Table of Contents

List of Tables

List of Figures

Abstract

MAKING SHAPES FOLDABLE

Zhonghua Xi, PhD

George Mason University, 2017

Dissertation Director: Dr. Jyh-Ming Lien

Recent advances in robotics engineering and material science accelerate the development of self-folding machines, the robots that can fold themselves from flat materials to functional 3D shapes. However, designing such self-folding machines remains extremely challenging. First, finding a 2D (flat) structure that can be folded back to the original 3D shape in nontrivial especially for non-convex shapes. Furthermore, whether there exists a folding motion that continuously transforms the foldable object from one state to another without self-intersection, is one of the major concerns but rarely explored area in self-folding robots. In this dissertation, I study both unfolding and folding problems for two types of foldable objects: rigid origami and nets of Polyhedra. I make three main contributions throughout the dissertation: 1) Consider motion in foldability optimization when designing foldable objects; 2) Make both unfolding and folding easier for the machine (algorithm) and human folders via a new geometric data structure and a new foldability-aware segment strategy; 3) Propose a novel approach to compress an object with thick surface material to its most compact form via stacking. This super compressed form enables the manufacturing (such as 3D-printing) and transportation of large object in a significantly smaller space.

Chapter 1: Introduction

1.1 From Paper Folding to Self-Folding Machines

Paper folding, also known as Origami, is an ancient Japanese art. As a hobby, Origami brings the joy to both children and adults for over a century. Many of us folded the classic paper crane sometime in our life. As a mathematical model, Origami inspires innovations in a wide range of domain including but not limit to maps, shopping bags, umbrellas that are used in our daily life, medical devices for minimally invasive procedures and foldable solar panels on the satellite. Recent advances in robotics engineering and material science have enabled the development of self-folding machines which are the robots that can fold itself from a flat sheet to one or more 3D target shapes to perform tasks. Rigid origami and the nets of polyhedra are two most widely used models for such foldable objects. Though there are some existing works on generating non-overlap unfoldings for a given polyhedron, they have the following issues: 1) No motion is considered when finding nets which is very important for robots to fold themselves. 2) Unoptimized nets are hard to fold for both humans and robots. 3) The foldable structures are usually manufactured in 2D, which require much larger spaces than that of the folded 3D structures. 4) Thickness of the material is often not considered. Our goal it to provide a computational framework to design (self-)foldable structures, we will address all the above issues throughout this dissertation.

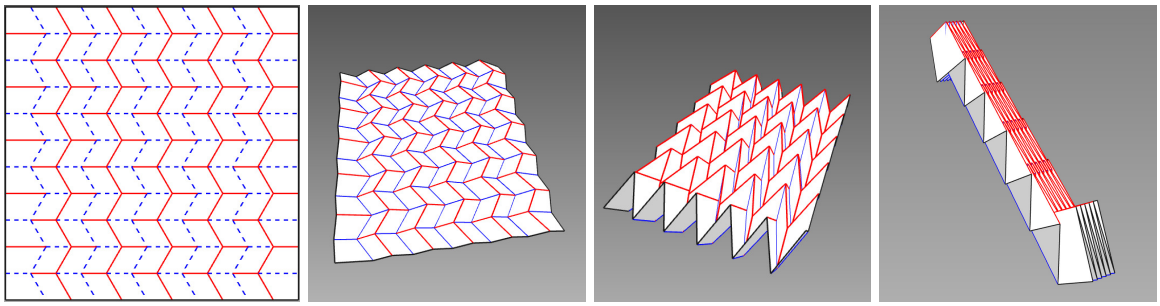


Figure 1.1: Folding process of a 11×11 Miura crease pattern.

1.2 Foldable Objects

1.2.1 Rigid Origami

Rigid origami has been a fundamental model in many self-folding machines [4] that are usually composed of mechanical linkage of flat rigid sheets joined by hinges, such as the micro-thick folding actuators [5]. In the past, people have enjoyed many practical uses of rigid origami, ranging from folding maps and airbags to packing large solar panel arrays for space satellites and folding space telescope. In the near future, rigid origami will take the form of self-folding machines and provide much broader applications, such as in minimally invasive surgery, where there is a need for very small devices that can be deployed inside the body to manipulate tissue [6]. Examples that illustrate the ability of transforming rigid origami from a shape to another can be found in Fig. 1.1, where a large flat sheet can be folded into a compact stick. A key issue in designing rigid origami is *foldability* that determines if one can fold a given origami form one state to another. It is known to the community that, given a crease pattern and a rigid goal configuration, the existence of *continuous rigid folding motion* is not guaranteed in general [7]. Unfortunately, there is no known criteria for determining whether a crease pattern or its tessellation can be folded between two rigid configurations without violating the rigidity constraint. In practice, when a crease pattern is designed, it usually requires its designer to create a physical copy and then verify that a rigid folding motion does exist to bring the crease pattern to a rigid goal

configuration. This process can often be costly and time consuming. Our ideas for addressing rigid foldability issues include: *adaptive randomized search* and *folding path reuse*. Specifically, we propose a deformation bounded folding planner that can ensure the rigidity of the origami during continuous folding motions; such planning has not been achieved before in the community. Given a tessellation formed with repetitive crease patterns, we further take advantage of its symmetry to reduce the degrees of freedom (DOF). For rigid origami with implicit folding orders, We propose a sampling-based motion planner that generates configurations using only a small set of folding angles, such as those found in the initial and final configurations and some commonly used angles such as $\frac{\pi}{2}$ and π. Given the simplicity of the proposed method, it provides many advantages comparing with a strategy that samples from the continuous space [8, 9]. First, our method can find more valid configurations in shorter sampling time. Second, our method can quickly discover *implicit folding order* that provides critical information to guide the folding process of many origami. It should be noted that, finding the implicit folding order, that requires the crease lines to be folded in a very specific order, can be viewed as the notorious "narrow passage problem" in sampling-based motion planners. Finally, contrary to the existing methods that only report a single folding path, our method can provide multiple folding paths in different homotopic classes.

1.2.2 Nets of Polyhedra

Making 3D shape from planar sheets is an ancient practice with many new applications, ranging from personal fabrication of customized items [10], which is fueled by the recent maker movement, to design of specialized instruments in self-folding machines [4] mostly due to the advances in active materials. One of the prevailing methods for creating 3D objects from planar materials is "unfolding and folding" [11]. Unfolding involves cutting a given polyhedral mesh into surface patches and then flattening them. To ensure that a surface patch can be flattened, existing methods either approximate the patch by developable surfaces or ensure that the patch forms a *net*, i.e., a patch that can be cut and flattened by

rotating its facets along one of the incident edges without overlapping with other facets [3]. The flattened patches are then cut out of planar materials and folded back to 3D.

Either cutting a polyhedral mesh into nets or approximating with developable surfaces, segmentation of the mesh (a process of breaking a mesh into multiple components) is usually involved; either before the unfolding algorithm is applied or as a product of the unfolding algorithms. Segmentation before unfolding is used as a preprocessing step to provide simplicity, as well as semantics [12, 13]. As shown in Fig. 1.2, segmentation is also a common technique used by paper craft designers. In the literature, shape segmentation is usually done without considering foldability [14, 15]. Consequently, surface patches produced by shape segmentation may still be cut into multiple nets or approximated by multiple developable surfaces which lose the semantic meaning and make folding and assembly less intuitive thus time consuming.

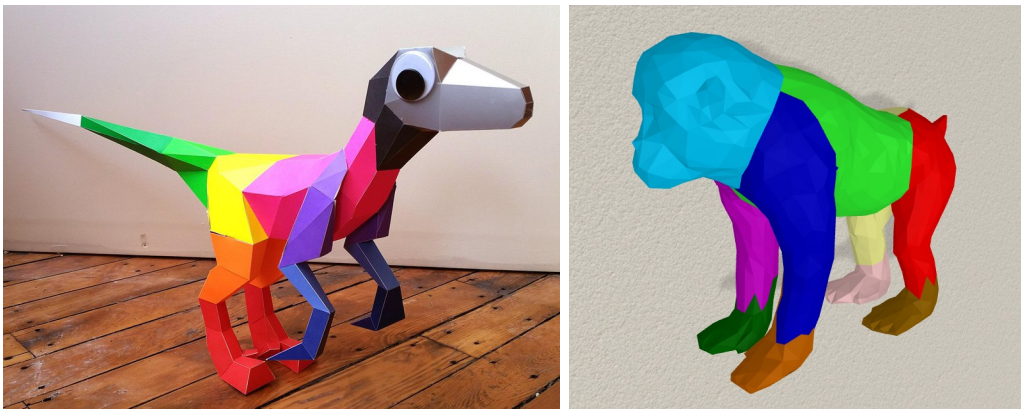


Figure 1.2: **Left:** A commercial paper craft designed by KitRex [1] that shows segmented parts with anatomic meanings. **Right:** Segmentation results of the Monkey model generated by the proposed method.

Segmentation can also be produced in order to avoid overlapping in the nets [16]. However, these nets often provide little shape information. In both scenarios, segmentation and unfolding operations has been decoupled. We propose a strategy that produces polyhedral

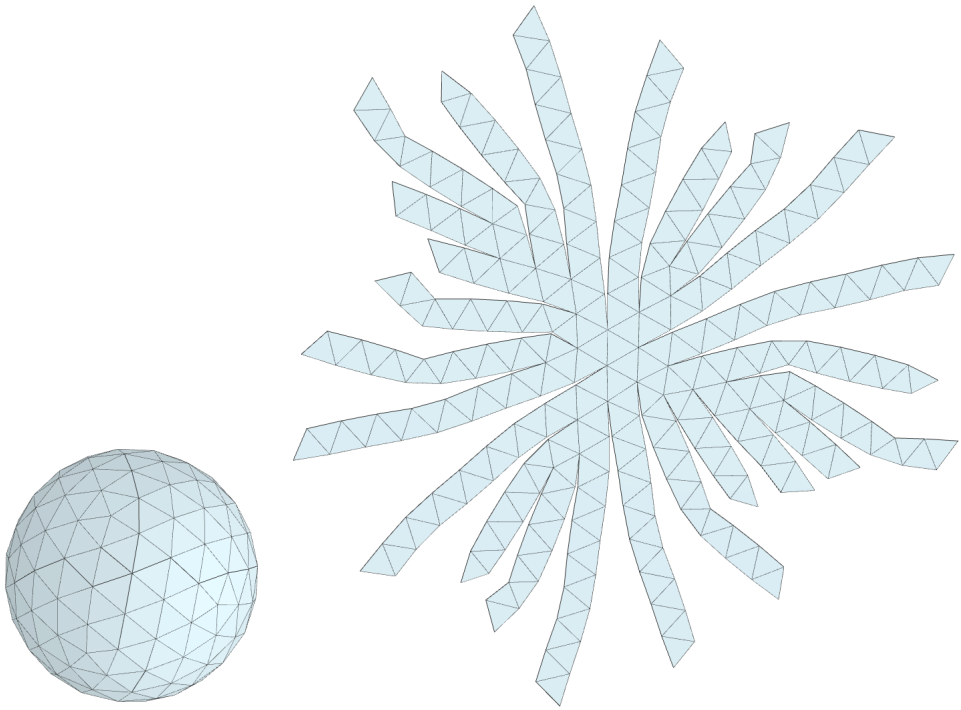
nets by tightly coupling the edge unfolding and surface segmentation operations. Our objective is to algorithmically produce nets that resemble carefully designed paper craft such as those shown in Fig. 1.2. We show that the proposed method naturally provides semantic segmentation of the input mesh by unfolding the entire mesh multiple times. Even though most likely, all of these unfoldings will contain overlaps, the proposed method learns from these failures and identifies parts that may be unfolded into valid nets. Existing shape segmentation methods rely heavily on shape features, such as curvature and geodesic distance. On the contrary, the proposed method creates the segmentation directly from information obtained from edge unfolding, therefore, ensures that every component in the segmentation can be unfolded into *a single net* and maintain its semantics.

1.3 The Unfolding Problem

For a given 3D polyhedron, finding one or a set of 2D representations whose folded states can (approximately) reconstruct the original polyhedron is called the unfolding problem. Each 2D component should not contain self-overlapping such that it can be physically realized. These representations include but not limit to: cones and planes [12], generalized cylinders [13], strips [17], crease patterns [18] and nets of polyhedra [2, 3, 16, 19]. In this dissertation, we focus on the unfolding the polyhedra into nets, mainly for the following reasons: 1) the folded shape is exactly the same as the original shape, 2) ease of fabrication 3) can be easily folded as by a human folder or as a self-folding machine. In Fig. 1.3, we show an example of the unfolding problem.

1.4 The Folding Problem

Once we have a 2D representation of the 3D shape, the next question we would like to ask is whether there exists a folding motion that transforms the 2D shape to the original 3D shape? And if so, how can we find a feasible folding path efficiently. This is called the continuous folding problem which is critical for realization a physical copy of the self-folding



(a) A sphere mesh (b) A net of the sphere mesh

Figure 1.3: A sphere mesh and its net.

machine that can fold itself without self-intersection. We show an example of the folding process of a net of a sphere mesh in Fig. 1.4. However, this is problem is usually ignored when solving the unfolding problem. In this dissertation, we tightly couple the unfolding problem and the continuous folding problem and present methods for solving both problems for rigid origami and nets of polyhedra.

1.5 Making Shapes Foldable

Throughout this dissertation, we propose several methods that efficiently unfold the given polyhedron into one or a small set of nets and the motion that transforms the net(s) back to the original shape. In Chapter 4, we present a method that simultaneously segments

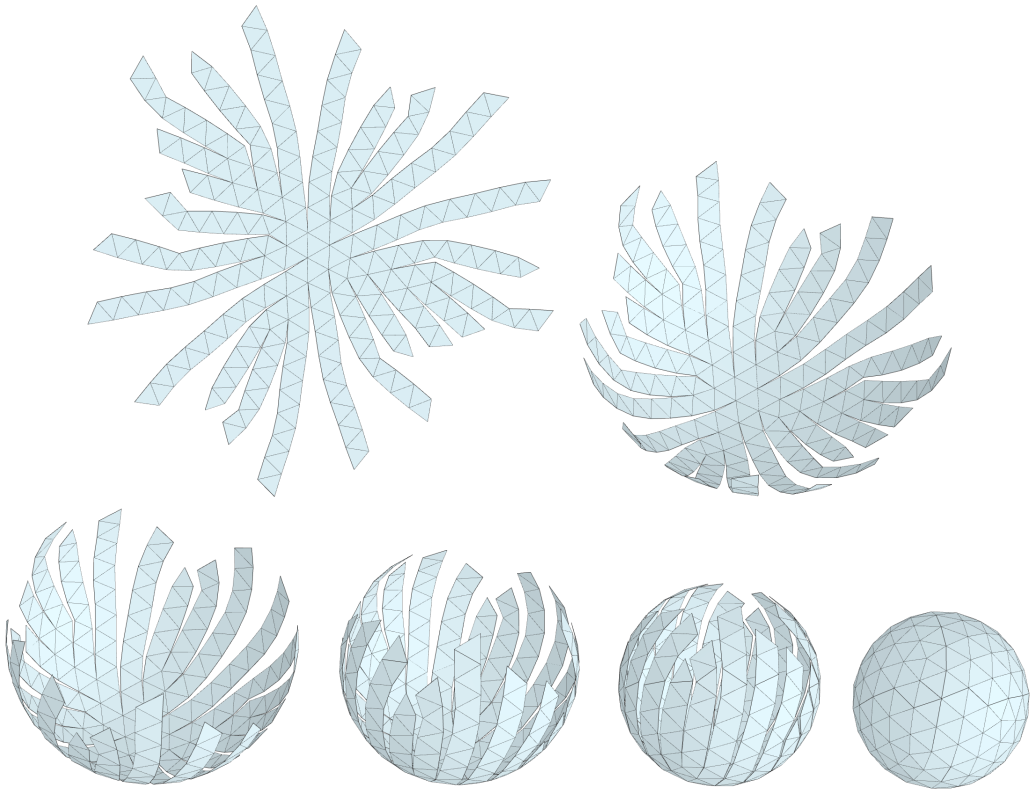


Figure 1.4: The folding process of a net of a sphere mesh.

and unfolds a non-convex shapes into a few nearly convex components such that each components can be easily unfolded. We propose a new geometric data structure called disjoint convex shell (DC-shell) in Chapter 5. A DC-shell consists a set of interior disjoint convex shapes that collectively approximate the original shape, such that each convex component can be easily unfolded to a net.

1.6 Fold as Compact as Possible

In Chapter 6, we propose a novel approach to make shapes foldable by approximating them with the surface voxelization. We also find the compactest folded state, we call it stacking, when using non-zero thickness material of their surface voxelization, then the approximated shape can be obtained by unfolding the stacking. From Fig. 1.3, we can see that the 2D

net have a significantly larger dimension then the folded 3D shape. Unlike the unfolding problem, the stacking found by the proposed method is the compactest folded state of the original 3D shape. The dimension can be significantly smaller than the original 3D shape, this great advantage enables us to fabricate (e.g. 3D-printed) a large 3D shape which can not be fit into the limited workspace. Our experimental results show that we can manufacture a model whose each dimension is 8x larger than the 3D-printer.

1.7 Organization

In Chapter 2, I introduce the rigid origami and polyhedra unfolding and briefly review the current literature as background material to help the readers better understand the foundations for the rest of the dissertation. Chapter 3 focuses on rigid origami, we discuss the modeling of the rigid origami and propose a randomized search algorithm to find feasible folding paths for rigid origami with closure constraints. Chapter 4 discusses the polyhedra unfolding problem in depth and various approaches used in pursuing this problem. Chapter 5 discusses a new geometric structure called disjoint convex shell or simply DC-shell. A DC-shell of a polyhedron is a set of pairwise interior disjoint convex objects that collectively approximate the given polyhedron. We show the advantages of DC-shell in both unfolding, folding and fabrication. Chapter 6 discusses the compact folding problem, we propose a novel approach to fold a 3D model into its compactest state using a technique we called stacking. The final chapter summaries my contributions to the origami folding/unfolding community and discusses some potential future works.

Chapter 2: Background

2.1 Rigid Origami

2.1.1 Preliminary

Crease Pattern

Crease patterns are widely used in the literature to represent the rigid origami model. A crease pattern is a straight-edged graph embedded in the plane. An edge of this graph correspond to the location of a crease line in an unfolded sheet of paper. A crease can be either *mountain folded* or *valley folded*. A mountain fold forms a convex crease at top with the paper beside the crease folded down. On the other hand, a valley fold forms a concave crease with both sides folded up. An example of mountain and valley folds is shown in Fig. 2.1.

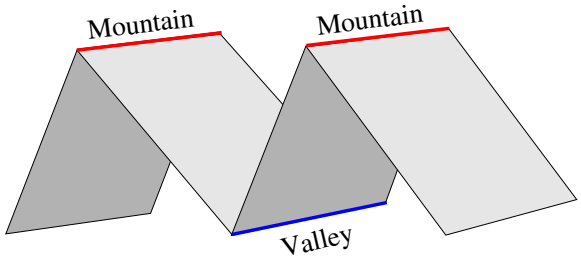


Figure 2.1: A crease can be folded as either a mountain fold (in red) or a valley fold (in blue).

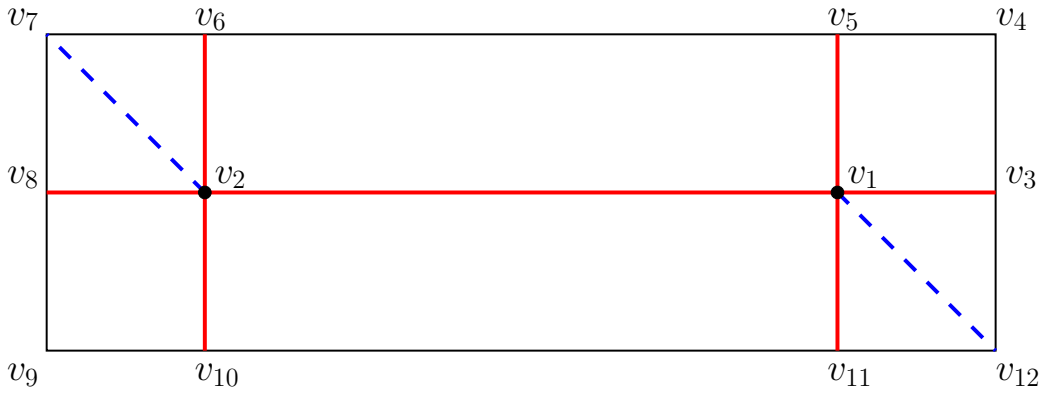


Figure 2.2: An example of multi-vertex crease pattern. Mountain creases are shown as solid lines in red, valley creases are show as dashed lines in blue.

Vertices in Crease Pattern

Vertices in the crease pattern can be categorized into two groups: real vertices and virtual vertices. Vertices on the boundary of the paper are considered as virtual vertices and they will not act as vertices for the purpose of our results. All other vertices on the paper are considered as real vertices. For example, vertices v_1 and v_2 in Fig 2.2 are real vertices and all the other vertices are virtual vertices.

Crease Lines and Faces

We use $l_{(i,j)}$ to denote a crease line that connects real vertex v_i and vertex v_j which can be either real or virtual, and we use $\rho_{(i,j)}$ to denote the folding angle of $l_{(i,j)}$. We use $F_{(i,j,...)}$ to refer the face that is on the left of $\overrightarrow{l_{(i,j)}}$ (e.g., $F_{(1,3,..)}$ is refer to $F_{(1,3,4,5)}$ in Fig 2.2). For real vertex v_i, we sort the crease lines $l_{(i,j_t)}$ in the order of the *plane angle* $\alpha_{(i,j)}$ which is the angle between x-axis and $\overrightarrow{l_{(i,j)}}$. We use c_i to denote the number of crease lines that are connected to v_i. For instance, in Fig 2.2, for real vertex v_1, $c_1 = 5$, the sorted crease lines are $\{l_{(1,3)}, l_{(1,5)}, l_{(1,2)}, l_{(1,11)}, l_{(1,12)}\}$.

Configuration

We use the folding angles of all crease lines as variables to represent the configuration of an origami model. More specifically, we define a configuration $\mathcal{C} = \{\rho_{(i_1,j_1)}, \rho_{(i_2,j_2)}, \cdots, \rho_{(i_n,j_n)}\}$ for an origami with n crease lines, where $\rho_{(i_k,j_k)}$ is the dihedral angle of two faces that are connected by the crease line $l_{(i_k,j_k)}$ on the folded shape. In this disseration, ρ is bounded in [-π, π] for origami, otherwise adjacent faces will penetrate each other during the folding process. In the real world, the range of ρ is further limited by the material (e.g., $\pi/2$ for DE material in [20]). Given the limitation, we are able to simulate the most compact shape when folding is done with real material. Given a crease pattern, the shape of the folded origami can be determined by the configuration $\mathcal{C}$ instantaneously, however, whether there exists a folding motion that continuously transform the crease pattern to the folded shape remains unknown.

Foldable and Feasible Configurations

Given a configuration $\mathcal{C} = \{\rho_{(i_1,j_1)}, \rho_{(i_2,j_2)}, \cdots, \rho_{(i_n,j_n)}\}$, we can classify $\mathcal{C}$ according to its *foldability* and *feasibility*. First, let v_i be a real vertex in a foldable multi-vertex crease pattern and let B_i be the 4×4 matrix which translates a point in $\Re^3$ by v_i. For crease line $l_{(i,j)}$, let $A_{(i,j)}$ be the matrix in homogeneous coordinates which rotates the xy-plane by plane angle $\alpha_{(i,j)}$ (the angle needed to rotate in CCW that can make positive x-axis overlap with the crease line $\overrightarrow{l_{(i,j)}}$), and let $C_{(i,j)}$ be the matrix in homogeneous coordinates which rotates by folding angle $\rho_{(i,j)}$ in the yz-plane. Then the folding matrix for the crease line $l_{(i,j)}$ around v_i will be $\chi_{((i,j),i)} = B_i A_{(i,j)} C_{(i,j)} A_{(i,j)}^{-1} B_i^{-1}$. If we pick $F_{(i,j_{c_i},\ldots)}$ (where c_i is the number of crease lines around v_i) as F_0 which will be fixed it in the xy-plane during folding and multiply the folding matrices though all crease lines that are around v_i in order of their plane angles $\alpha_{(i,j)}$, then

$$\prod_{t=1}^{c_i} \chi_{((i,j_t),i)} = I \tag{2.1}$$

These necessary conditions of foldability for multi-vertex rigid origami were first discovered by Balcastro and Hull in 2002 [21].

There are several properties that the folded paper should have:

1. unstretchable,
2. flat (planar) for all faces,
3. free of self-intersection,

A foldable configuration $\mathcal{C}_{foldable}$ only guarantees the first two properties. In order to have a valid configuration $\mathcal{C}$ that satisfy all three of these properties, we need to check if $\mathcal{C}$ is free of self-intersection. In order to do so, we will need a folding map for each face. A folding map is a function that map a point in $\Re^2$ to the corresponding point of folded state for a given foldable configuration $\mathcal{C}_{foldable}$ in $\Re^3$. With the folding map for all faces, we can fold the paper to the foldable configuration $\mathcal{C}_{foldable}$ and perform collision detection to check the feasibility of $\mathcal{C}_{foldable}$.

Folding Map

Let F_0 be an arbitrary face that will be fixed in the xy-plane during the entire folding process, and given another face $F_{(i_p,j_p,...)}$, let γ be any *vertex-avoiding path* starting from a point in F_0 and ending at a point in $F_{(i_p,j_p,...)}$ by crossing some crease lines. We say that a path is vertex-avoiding if it does not intersect with any vertices. Let the crease lines that γ crossed be, in order, $l_{(i_1,j_1)}, ..., l_{(i_p,j_p)}$. Then the folding map for $(x, y) \in F_{(i_p,j_p,...)}$ is:

$$f(x, y) = f(x, y, 0, 1) = \prod_{k=1}^{p} \chi_{((i_k,j_k),i_k)}(x, y, 0, 1) \tag{2.2}$$

Note that the folding map is independent of the path γ. This means no matter which path we choose, the folding map remains the same if C is foldable configuration. This

property was first proved in [21]. Thus, we can pick the shortest path γ from F_0 to $F_{(i_p,j_p,\ldots)}$ using depth-first-search and compute the production of rotation matrices as the folding map for $F_{(i_p,j_p,\ldots)}$.

2.1.2 Planning and Simulating Origami Motion

In 1996, Miyazaki et al. [22] simulate origami folding by a sequence of simple folding steps, including bending, folding up, and tucking in. It is easy to reconstruct an animation from a sheet of paper to the final model. However, the simplicity of folding steps limits the types of origami models that could be represented in the system. Consequently, this method is not suitable for many complex origami models whose folding process cannot be represented as simple folding steps such as the Miura pattern shown in Fig. 1.1. Song et al. [8] presented a probabilistic-roadmap-method (PRM) based framework for studying folding motion. However, their kinematic representation of origami is a *tree-structure model* whose folding angle of each crease line is independent of other crease lines. Although tree-structure model greatly simplifies the folding map that can be easily defined along the path from base to each face, this model is not applicable to represent the majority of the origami, such as the one shown in Fig. 1.1, due to their *closure constraints*. Balkcom [23] proposed a simulation method based on the ideas of virtual cutting and combination of forward and inverse kinematics using a rigid origami model. Although this approach is computational efficient, it cannot guarantee the correct mountain-valley assignment for each crease, i.e., a mountain fold can become a valley fold or vice versa. More recently, Tachi [24] proposed an interactive simulator for rigid origami model (known as Rigid Origami Simulator (ROS)) which generates folding motion of origami by calculating the trajectory by projection to the constrained space based on rigid origami model, global self-intersection avoidance and stacking order problems are not considered in his work. Perhaps the work closest to our approach proposed in this disseration is by An et al. [5]. They proposed a new type of self-reconfiguration system called self-folding sheet. They first construct the corresponding folded state for a given crease pattern and angle assignment then continuously unfold the

paper using local repulsive energies (via a modification of ROS [24]). By reversing the unfolding sequence, they obtained the path starting from a flat sheet and ending with the desired folded state.

2.1.3 Planning under Closure Constraints

Although there exists little work on origami motion planning, there have been many methods proposed to plan motion for articulated robots under closed-chain constraints [25, 26, 27, 28]. Interestingly, we see many similar ideas used in both closed-chain systems and origami folding. For example, gradient decent was used by [24] for rigid origami simulation and by [25] for generating valid configuration of a closed-chain system. Another example is inverse kinematics, which plays the central role both in Balkcom's simulator [23] and in constructing the so-called *kinematic roadmap* [27, 29] for capturing the topology of free configuration space. Tang et al. [30] proposed an efficient sampling-based planner for spatially constrained systems. By sampling in the reachable distance space in which all configurations lie in the set of constraint-satisfying subspaces and using a local planner, they can significantly reduce the computation time for finding a path.

2.2 Polyhedra Unfolding

2.2.1 Preliminary

Nets of Polyhedra

Let $\mathcal{M}$ be a polyhedral mesh, the graph of mesh is defined as $G(\mathcal{M}) = (V, E)$, where $V = \{\text{vertices of } \mathcal{M}\}$ and $E = \{\text{edges of } \mathcal{M}\}$. The dual graph $G'(\mathcal{M})$ of $G(\mathcal{M})$ is defined as $G'(\mathcal{M}) = (V', E')$, where $V' = \{\text{faces of } \mathcal{M}\}$ and $E' = \{(u, v)\}$, where u, v are faces and share an edge in $\mathcal{M}$.

The unfolding can be obtained by finding a spanning tree of the dual graph $G'(\mathcal{M})$ [3]. Folding edges will be those edges that are crossed by dual edges in the spanning tree, all rest of the edges are cut edges that will be cut in order to unfold the mesh. The an unfolding

of a polyhedron does not contain overlaps then it is called the net of the polyhedron. An example of dual graph of a mesh and its unfolding is shown in Fig.2.3

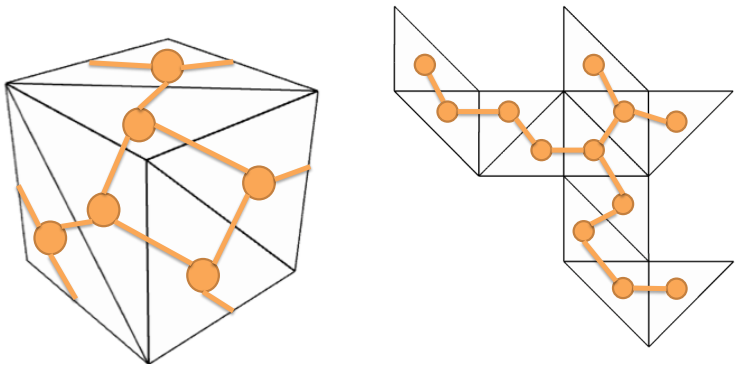


Figure 2.3: **Left:** Mesh and its dual graph, **Right:** Spanning tree of the dual graph and the unfolding of the mesh.

Configuration

We model the net of polyhedron as a multi-link tree-structure articulated robot. Similar to rigid origami, we use the folding angles of all fold edges as variables to represent the configuration of a net. More specifically, we define a configuration $\mathcal{C} = \{\rho_{i_1}, \rho_{i_2}, \cdots, \rho_{i_n}\}$ for a net with n fold edges, where ρ_i is the dihedral angle of two faces that connected by the fold edge e_i on the folded shape.

Folding Map

Let F_{i_0} be an arbitrary face that will be fixed in the xy-plane during the entire folding process, and given another face F_{i_p}, let γ be a path starting from a point in F_0 and ending at a point in F_{i_p} by crossing some fold edges. Let the fold edges that γ crossed be, in order, $\{e_{i_1}, \cdots, e_{i_p}\}$, and let the faces γ entered be, in order, $\{F_{i_0}, F_{i_1}, \cdots, F_{i_p}\}$. Then the folding map for $(x, y) \in F_{i_k}$ can be defined as,

$$f_{i_k}(x,y) = f_{i_k}(x,y,0,1) = \begin{cases} I_{4\times4}, & \text{if } k = 0 \\ f_{i_{k-1}} R(\vec{e'_{i_p}}, \rho_{i_p}), & \text{otherwise} \end{cases} \tag{2.3}$$

where $R(\vec{e}, \rho)$ is a rotation matrix that rigidly transforms a point by rotating around $\vec{e}$ for ρ and $\vec{e'_{i_p}}$ is the folded state of edge e_{i_p}.

2.2.2 Edge Unfolding

Mathematicians spend centuries in answering a question: given a polyhedron, is it always possible to unfold the polyhedron by cutting on the surface of it, such that the unfolding of the polyhedron does not contain overlapping? When cuts are restricted only to the edges of polyhedra, it becomes an edge-unfolding problem. For edge-unfolding, counterexamples were found for non-convex polyhedra; for convex polyhedra, though promising, it still remains open [31]. Later, heuristic methods were proposed in the literature for unfolding convex polyhedra to nets [3]. However, it becomes much harder to generate a single net for non-convex shapes. Both [2, 16] generate more than one connected components for complex non-convex shapes to avoid overlapping. The former one splits the unfolding when overlaps were detected while the later one first splits the mesh into multiple pieces then tries to merge them into one piece. All aforementioned works generate nets as final results, however, whether there exists a continuous folding motion that transforms the net back to its original shape is not considered in their works.

2.2.3 Paper Crafting via Shape Segmentation

Unfolding a non-convex polyhedron into a single connected component is hard but not necessary for certain applications, such as paper crafting: making 3D models from flat sheets

of paper. For paper crafting, shape segmentation techniques were employed to simultaneously decompose and approximate the mesh into smaller pieces [12, 13, 17] such that the unfolding problem becomes solvable and the approximated 3D model can be obtained by assembling the folded shapes together. These approaches decompose the mesh into a few patches and approximate each patch with a strip, a generalized cylinder or a developable surface. Common drawbacks of these methods are that they could generate an arbitrary number of pieces and the cuts can be at arbitrary locations on the mesh which make assembling much harder and less fun, also the approximation ability is limited. Another category of shape decomposition method worth noting is called Nearly Convex Decomposition (NCD) [32, 33, 34], which segments a mesh into a controllable number (usually small) of part-aware components that are nearly convex. Mesh convexification shows great advantages in unfolding and continuous folding, and we can obtain either exactly the same model as the original one by assembling folded nearly convex patches, or an approximated model (with bounded error) by folding the nets generated from the convex hulls of those patches.

Chapter 3: Rigid Origami

3.1 FROG: A Randomized Path Planner for Rigid Origami

Similar to the problem faced in systems with closure constraints, traditional motion planners that perform local planning using linear interpolation usually fail to connect two seemingly nearby configurations. Moreover, we observe that, for rigid origami, whose folding angles are highly constrained by each other, its folding pathway has very distinct characteristics between the early folding stage and the rest of the folding process. That is, there are abundant valid configurations when the origami is still flat, however, once the folding process started, the folding pathway quickly becomes very narrow and highly non-linear due to the closure constraints. This difference can be observed from the smoothness of the trajectories shown in Fig. 3.1(b) and Fig. 3.1(c). The former is much smoother than the latter. As we will see later in this section, these observations play important roles in designing our randomized folding algorithm.

Finding a path in a highly constrained high-dimensional configuration space is always challenging. We propose FROG(Folding Rigid OriGami), a randomized path planner for rigid origami, to repetitively sample a configuration $\mathcal{C}_\tau$ randomly near he best configuration known to us $\mathcal{C}_\triangle$ so far, and use a non-linear optimization approach to find a valid configuration locally around $\mathcal{C}_\tau$. FROG only expands the closest configuration to the goal and uses an adaptive weight adjustment to balance between randomness and the desire to move towards to the goal. Details of the proposed method are described in Algorithm 1.

Algorithm 1 first initializes the planner by setting the $weight$ to W_0, and set the closest configuration $\mathcal{C}_\triangle$ to S. In each step, it samples a random configuration $\mathcal{C}_{rand}$ and find a direction $\overrightarrow{dir}$ by linearly combining $\mathcal{C}_{rand}$ and G with corresponding weights, $1 - weight$ and $weight$, respectively. Then, a new configuration $\mathcal{C}_\tau$ is created by moving $\mathcal{C}_\triangle$ forward

Algorithm 1 FROG

Input: Start configuration S, goal configuration G
Output: Foldable and feasible path from S to G

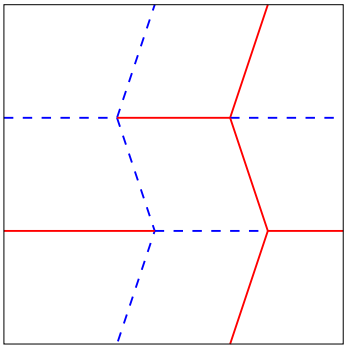
$weight \leftarrow W_0$
$\mathcal{C}_\triangle \leftarrow S$
while G not reached **do**
    $\mathcal{C}_{rand} \leftarrow$ a random configuration
    $\overrightarrow{dir} \leftarrow (1 - weight) \cdot \mathcal{C}_{rand} + weight \cdot G$
    $\mathcal{C}_\tau \leftarrow \mathcal{C}_\triangle + D \cdot \overrightarrow{dir}$
    $\mathcal{C} \leftarrow$ FindFoldable($\mathcal{C}_\tau$)
    if IsValid($\mathcal{C}$) and $\mathcal{C}$ is closer to G **then**
        $\mathcal{C}_\triangle \leftarrow \mathcal{C}$
        $weight \leftarrow weight + W_1$
    else
        $weight \leftarrow weight - W_2$
    end if
end while

distance D along $\overrightarrow{dir}$. However, even if D is a tiny number, the target configuration $\mathcal{C}_\tau$ is usually unfoldable. Thus, we introduced the function FindFoldable for finding a foldable configuration $\mathcal{C}$ around $\mathcal{C}_\tau$. If $\mathcal{C}$ is feasibly and it is closer to the goal G than $\mathcal{C}_\triangle$, Algorithm 1 replaces $\mathcal{C}_\triangle$ by $\mathcal{C}$. We use Euclidean distance to measure how far away two configurations are, however, other metric (e.g., 1-norm distance or infinity norm distance) can also be used. Algorithm 1 repeats this process until G is reached. The value of *weight* will be adjusted adaptively during the process.

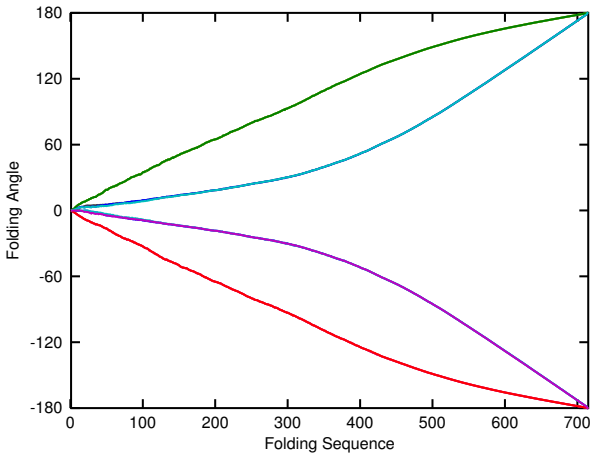
3.1.1 Finding Foldable Configuration

Non-linear optimization (NLOPT) is used to find a foldable configuration in function FindFoldable in Algorithm 1. Given a configuration $\mathcal{C}_\tau$, FindFoldable pushes $\mathcal{C}_\tau$ to a foldable configuration $\mathcal{C}$ near $\mathcal{C}_\tau$ by minimize the objective function shown in Eq. (3.1).

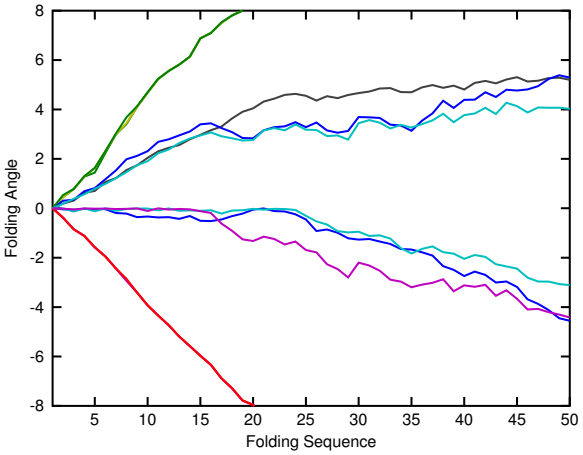
$$F(\mathcal{C}) = \max_{i=1}^{n_v} \left| \prod_{k=1}^{c_i} \chi((i, j_k), i) - I \right| , \tag{3.1}$$



(a) Crease pattern



(b) Folding path



(c) First 50 steps of the folding path

Figure 3.1: Miura crease pattern and the folding path of it found by the proposed method.

where n_v is the number of real vertices, c_i is the number of crease lines incident to the real vertex v_i, and, finally, (i, j_k) is the k-th crease lines around v_i.

More specifically, for a given real vertex v_i, we want the production of rotation matrices of crease lines around v_i to be as close to an identity matrix as possible. Since, in a foldable configuration, each real vertex in the crease pattern should be an identity matrix shown in Eq. (2.1). Special treatment for the stationary face F_0, which is fixed in the xy-plane, is required. The folding map of F_0 shown in (2.2) should always be an identity matrix regardless which closed vertex-avoiding loop γ is used for computing the folding map, otherwise F_0 will no longer stay in the xy-plane. Note that, NLOPT might still return an unfoldable configuration due to that maximum iteration has been exceeded or no foldable configuration exists around the configuration $\mathcal{C}_\tau$. These unfoldable configurations can be filtered out by performing the foldability check on the returned configuration.

3.1.2 Detecting Invalid Configuration

A foldable configuration might still be an invalid configuration due to self-intersection. Thus a collision checking is applied after the configuration $\mathcal{C}$ is returned by FindFoldable in Algorithm 1. Local intersection is avoided by bounding the folding angle in $[-\pi, \pi]$ for each crease line. Global intersection is avoided by applying collision detection between faces of the origami. In our implementation, we checking collision on all pairs of faces and we say an origami has self-intersection if penetration is detected while face overlapping is considered as valid.

3.1.3 Experimental Results

We show the crease pattens and their folded states used in our experimental in Fig. 3.2 and Fig. 3.3 respectively. And we show the running time to find feasible folding paths for each crease pattern in Table 3.1. From which we can see that the proposed algorithm efficiently finds folding paths for rigid origami.

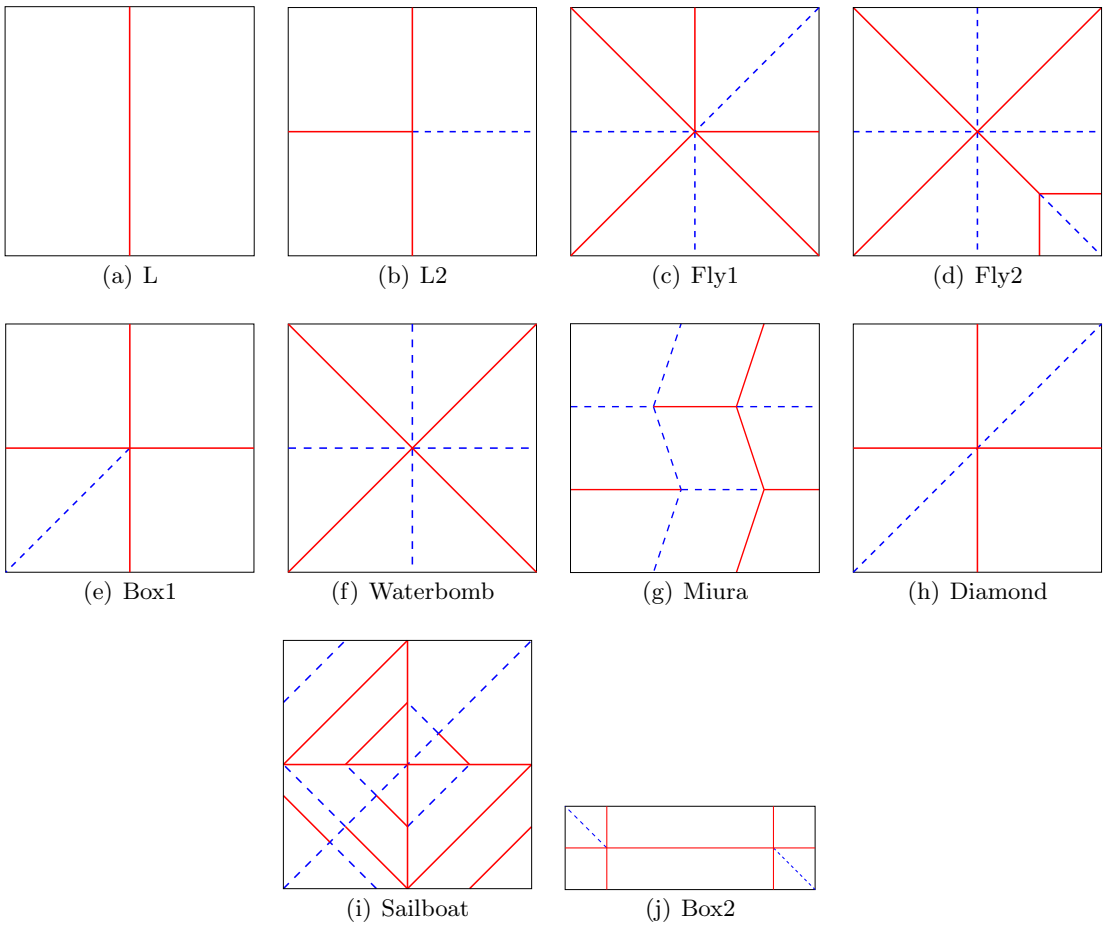
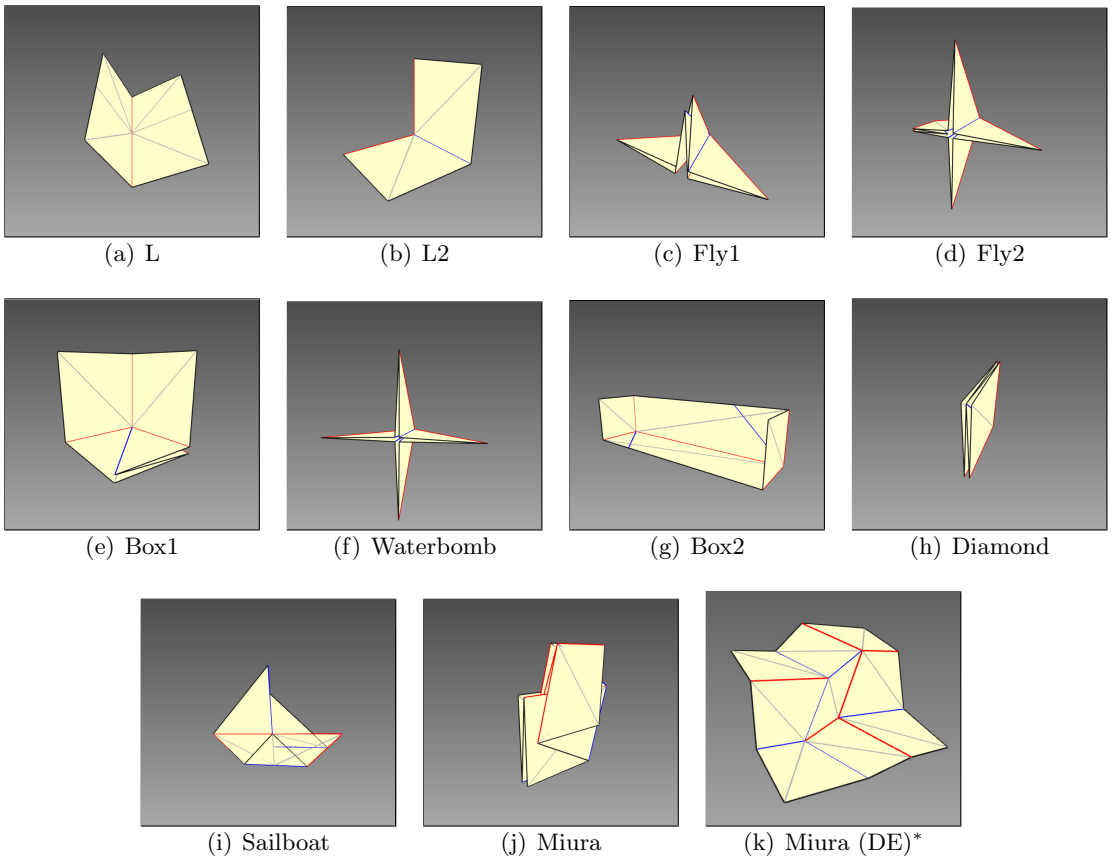


Figure 3.2: Crease patterns used in our experiments. Mountain creases are shown as solid lines in red, valley creases are show as dashed lines in blue.

3.2 MD-FROG: A Path Planner for Multi-DOF Rigid Origami

In this section, we present MD-FROG, a path planner for multi-DOF rigid origami. We say an origami is Multi-DOF if there exists a configuration that under which one or more crease lines can be folded/unfolded independently, i.e. its rigidity can be maintained without folding other crease lines.



(a) L (b) L2 (c) Fly1 (d) Fly2

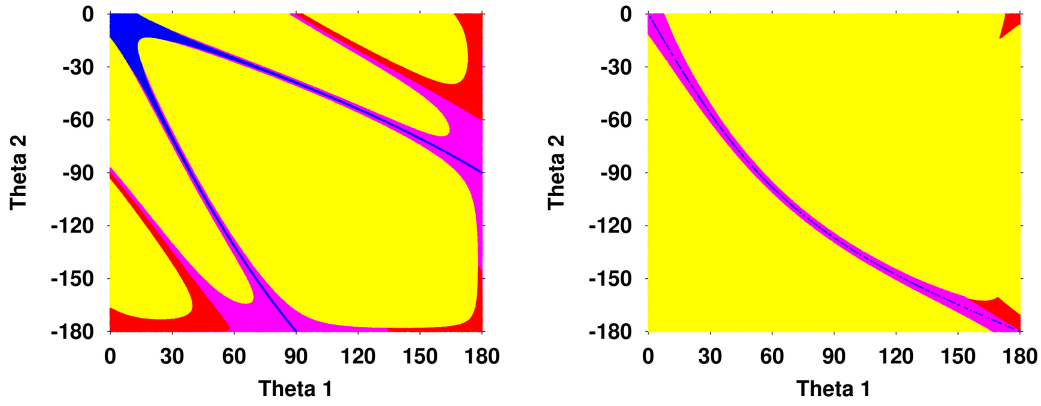
(e) Box1 (f) Waterbomb (g) Box2 (h) Diamond

(i) Sailboat (j) Miura (k) Miura (DE)*

Figure 3.3: Folded states of crease patterns shown in Fig. 3.2. Note that some of the models do not fold completely for the sake of better visualization.*Folding with DE material, which has a maximum folding angle of $\pi/2$.

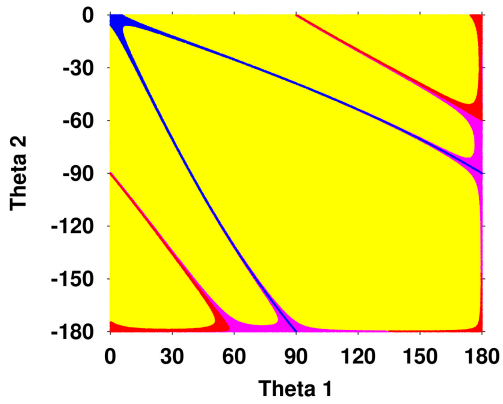
3.2.1 Sampling In Discrete Domain

Traditional sampling strategies have difficult to effectively generate valid samples in the configuration space for rigid origami with closure constraints even in lower dimensional space. Some crease patterns have been shown to be 1-DOF mechanism such as the Miura crease pattern [35] which means the valid configurations form a curve in the configuration space, thus the probability of a random configuration to be valid is zero. Although we could tolerant certain amount of deformation, the configuration space is still mostly occupied by "obstacles" as shown in Fig. 3.4, only 0.044% of the configuration space is valid under 0.1% deformation tolerance for the Miura crease pattern with number of crease lines reduced to

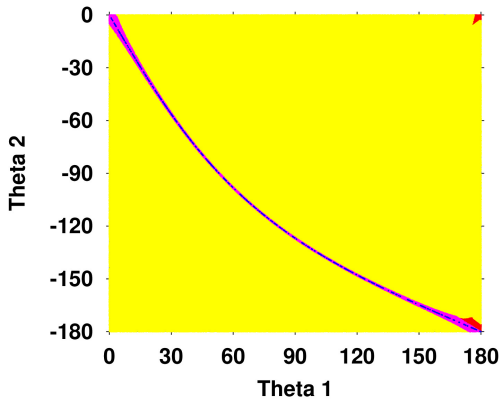


(a) Waterbomb 5% DT

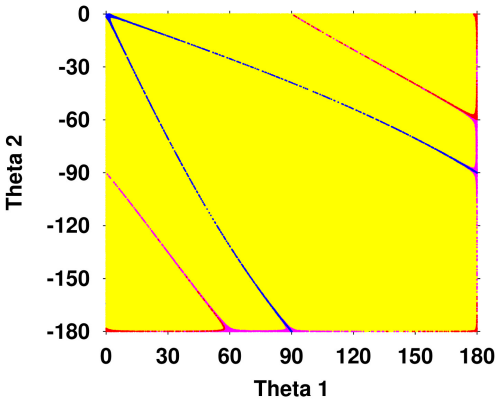
(b) Miura 5% DT



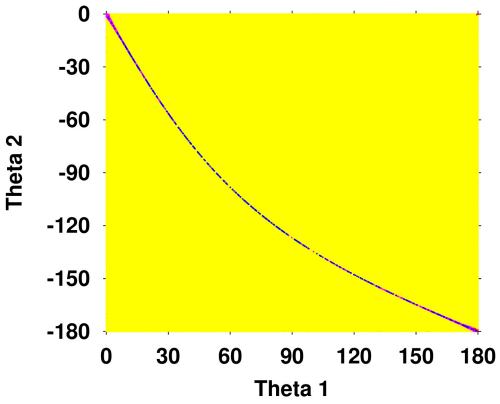
(c) Waterbomb 1% DT



(d) Miura 1% DT



(e) Waterbomb 0.1% DT



(f) Miura 0.1% DT

Figure 3.4: Random sample one million configurations uniformly for a Waterbomb crease pattern (Left) and a Miura crease pattern (Right) under different deformation tolerances (DT). Red: has self-intersection, invalid. Yellow: deformation is larger than tolerance, invalid. Magenta: within deformation tolerance but actual folding angles are different from assigned ones, invalid. Blue: valid.

Model	DOF	Time (s)	Sampled	Valid	Tested
L	2	0.004	110	110	156
L2	4	8.317	40323	787	2047635
$L2^2$	4	0.010	331	331	407
Box1	6	0.101	250	250	15181
Diamond	6	0.514	1074	390	65033
Waterbomb	8	0.613	731	359	54046
Fly1	8	0.631	605	421	53613
Box2	9	1.463	864	358	126885
Fly2	11	2.425	864	460	127064
Miura	12	13.564	1813	677	511050
Sailboat	34	83.923	6103	1970	1396032

(Note: $L2^2$ is L2 with an intermediate state defined by user)

Table 3.1: Running Time of Finding a Feasible Path.

2 by taking symmetry into consideration [36]. And situation will become even worse in higher dimensional configuration space.

To address this problem, instead of sampling in the continues domain with zero probability to generate a valid configuration, we propose the idea of sampling in the discrete domain. For a crease line with target folding angle ρ, we only sample the folding angle from its *important angle set*: $\{0, \pi, \rho\}$ which are corresponding to the flat state, the fully folded state and its target state. The total number of unique configurations for the origami with n crease lines is 3^n. For 1-DOF origami, usually it has only two continuous foldable configurations in the discrete domain which represent the initial state and the target state. And for Multi-DOF origami we expect to find more valid and continuous foldable configurations.

3.2.2 Connecting Two Valid Configurations

Given two valid configurations, it is usually unknown whether a rigid foldable and collision free path exist or not due to closure constraints which usually result in highly nonlinear path. In order to connect two configurations, we employ two connection methods: linear connection and nonlinear connection.

Linear connection An intuitive but turns out the most efficient way to connects two valid configurations is by linearly interpolating the intermediate configurations. Two configurations are rigid foldable to each other if all interpolated intermediate configurations are rigid foldable and collision free.

Nonlinear connection If linear connection failed to connect two configurations which means the path has to be nonlinear or even does not exist. We use a randomized search method proposed in [9] to connect two valid configurations, which could find a nonlinear, rigid foldable and collision free path.

3.2.3 Path Planning

We propose MD-FROG, a folding path planner for a Multi-DOF origami under the Lazy-PRM framework [37]. First, we sample configurations in the discrete domain and add valid configurations to the roadmap. We then connect all pairs of the configurations initially and add the edges to the roadmap. Then a graph query is answered to find a path from start node to target node. Connectivity checking will be applied only on the consecutive nodes in the path. If two nodes cannot be connected, i.e., they are not continuous foldable to each other, their corresponding edge is removed from the roadmap and a new path is extracted. We repeat this process until all edges that connect consecutive nodes in the path are validated. Finally, the rigid foldable and self-intersection free path is obtained by combining all the path segments.

Via intermediate configurations, we found alternative folding path for the Waterbomb crease pattern, folding path and folding process are shown in Fig. 3.6(b) and Fig. 3.6(d) respectively. As we can see from Fig. 3.6(b), the origami folds to goal state via an intermediate state and one of the path segments is linear.

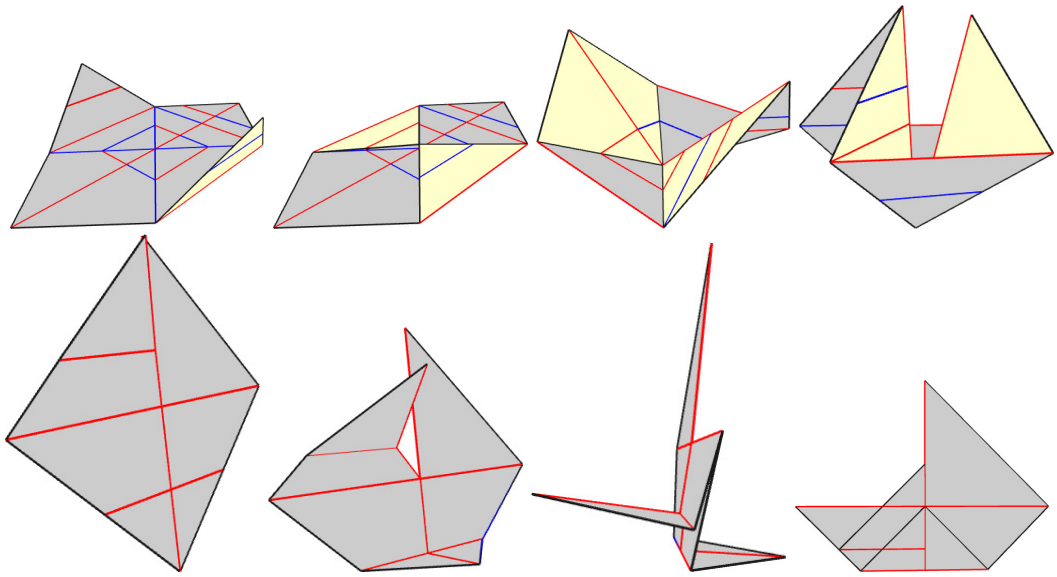


Figure 3.5: Folding sequences of a rigid sailboat origami produced by the proposed planner which configurations sampled in the discretized configuration space of the sailboat crease pattern.

3.2.4 Experimental Results

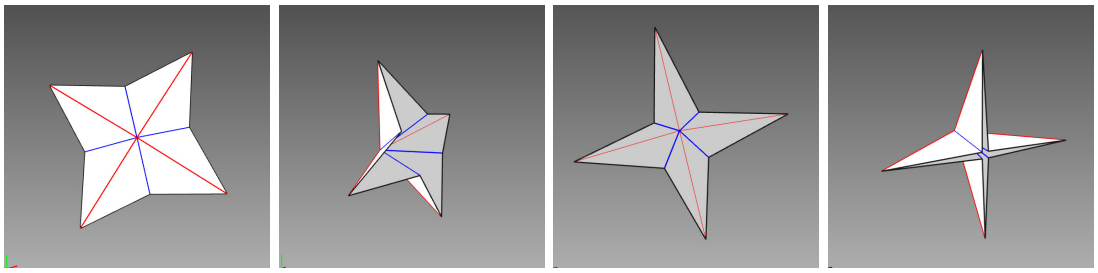
3.2.5 Continuous V.S. Discrete Sampling Strategy

In order to evaluate our method, we conduct an experiment on the crease patterns shown in Fig. 3.7. The number of crease lines n in the crease pattern we used are from 2 to 12 shown in Table 3.2, which equal to the dimensionality of the configuration space.

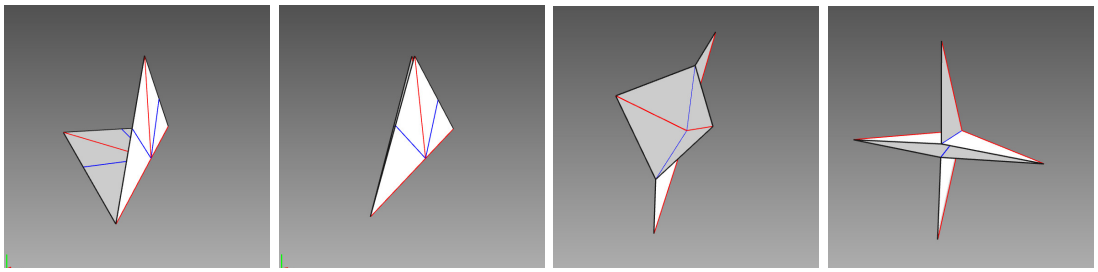
We uniformly sample one million random configurations in the configurations space and compare the number of valid samples and their running time.

Sampling in continuous domain As we can see from Table 3.2, even in lower dimensional space (e.g., 2D) it can generate only a few valid configurations, for Waterbomb and Miura crease pattern in 2D, the valid configurations are only about 1.02% and 0.13% respectively. With the increase of dimensionality, it failed to find any valid configuration even though the origami is Multi-DOF due to closure constraints.

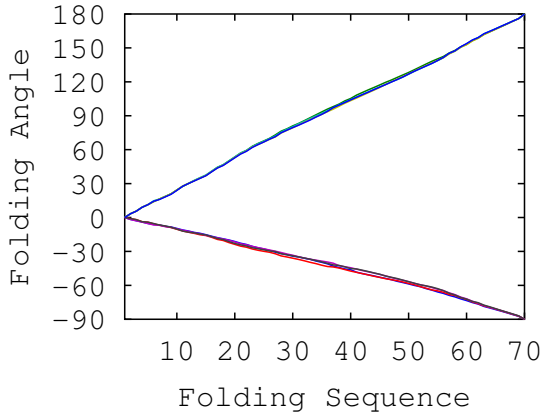
Sampling in discrete domain For the proposed method, folding angles are sampled only from each crease line's *important angle set*: $\{0, \pi, \rho\}$ as **Discrete3**. For comparison



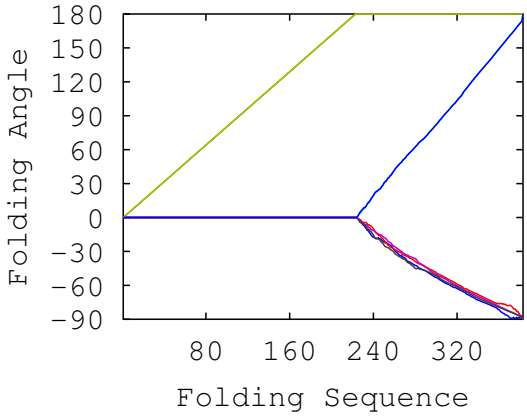
(a) Folding path found by FROG[9]



(b) Alternative path found by MD-FROG[36]

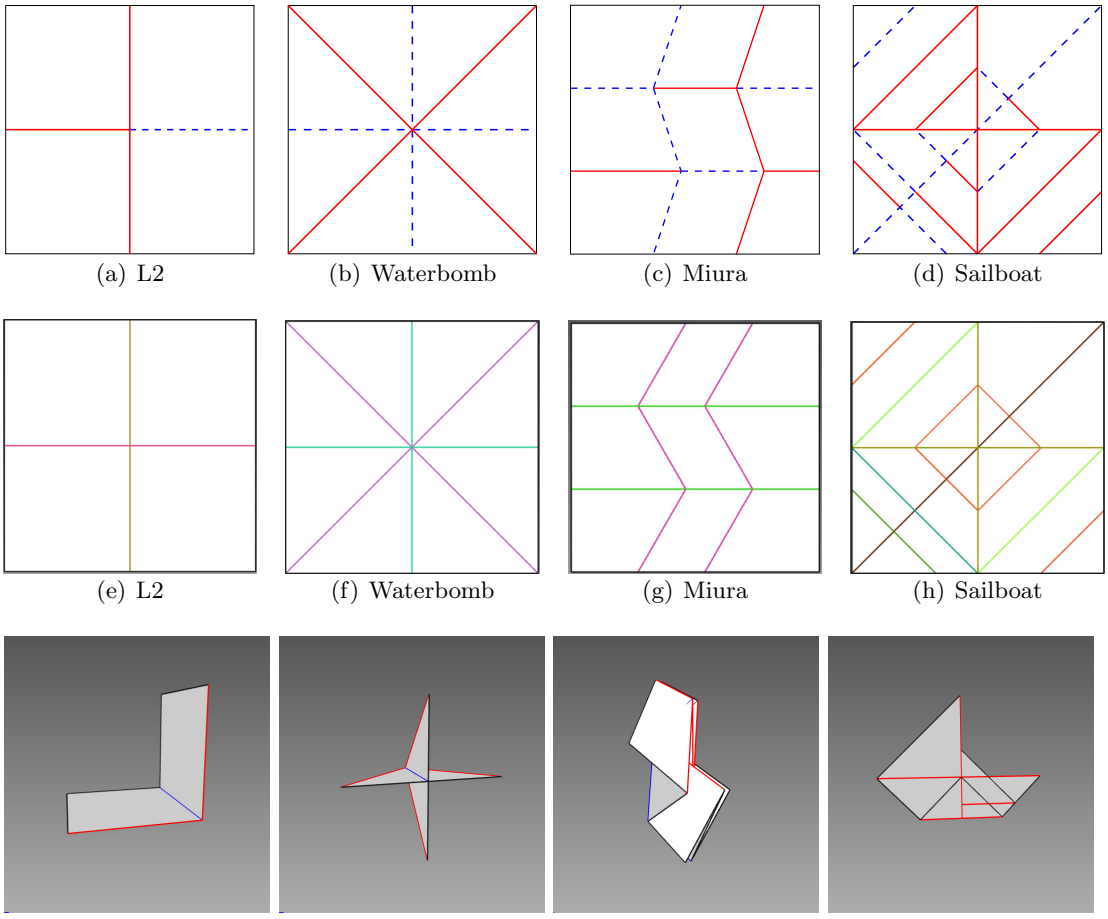


(c) Folding angle trajectories of (a)



(d) Folding angle trajectories of (b)

Figure 3.6: Valid states and folding process of a Waterbomb crease pattern.



(a) L2 (b) Waterbomb (c) Miura (d) Sailboat

(e) L2 (f) Waterbomb (g) Miura (h) Sailboat

(i) Target shapes

Figure 3.7: **Top**: Crease patterns used in our experiments. Mountain creases are shown as solid lines in red, valley creases are show as dashed lines in blue. **Middle**: Crease patterns with crease lines in groups. crease lines in the same group are shown in the same color. **Bottom**: Target shapes of above crease patterns.

we also sampled from another angle set: $\{0, \pi/2, \pi, \rho\}$ as **Discrete4**. From Table 3.2 we can see that, this strategy finds several intermediate configurations efficiently since we can filter out duplicated configurations in constant time using a hashtable, and effectively as we will see later those intermediate configurations are very important.

Model	n	Continuous		Discrete3		Discrete4	
		Valid	Time (s)	Valid	Time (s)	Valid	Time (s)
Waterbomb*	2	10161	12.10	5	0.35	6	0.42
Miura*	2	1305	17.48	3	0.36	3	0.43
L2	4	1	10.31	5	0.45	6	0.52
Sailboat*	6	0	33.70	48	0.57	118	0.74
Waterbomb	8	0	10.97	71	0.73	114	0.89
Miura	12	0	17.35	7	0.94	7	7.13

* indicates that symmetry property is used.

Table 3.2: Comparison Between Sampling Strategies.

In this experiment we show that sampling in discrete domain is a powerful strategy to generate valid samples for rigid origami with closure constraints. This strategy works even when the sampling domain is small (in our case only three values) and enables us to discover foldable states while sampling in continuous domain was not able to find any.

3.3 Reusing Folding Path

A tessellation is a type of crease pattern that can usually be viewed as an arrangement of smaller repetitive crease patterns. As a result, the degrees of freedom of a tessellation is usually very large (758 for a 12×22 Waterbomb and 1680 for a 24×24 Miura fold). Finding valid folding motion for such as tessellation can be extremely time consuming. In order to speed up the motion planner, we propose the idea of *crease group* and *essential vertex* by exploiting symmetry in the tessellation. Computation reuse is a widely used technique to

improve the performance of a robotic system [38]. We propose the idea of reusing folding path found on the *essential crease pattern* to fold large origami tessellation.

3.3.1 Crease Group and Essential Vertex

Given a large crease pattern (tessellation), crease lines can be gathered into groups naturally due to symmetry property. We say that a set of crease lines are in one crease group if the absolute value of their folding angles trace out the same folding trajectory. In Fig. 3.7 we can see that the absolute value of folding angles of 12 crease lines trace out only 2 trajectories. Given the crease groups, we define *essential vertices* as a set of real vertices whose incident crease lines collectively *cover* all the crease groups. The smallest essential vertices can be found by solving the *set covering problem*. An example of crease groups is shown in Fig. 3.8, in which crease lines belong to the same crease group are shown in the same color. From Fig. 3.8 we can see that the 3×3 Miura crease pattern has only two crease groups: all vertical crease lines are in one group and all horizontal crease lines are in another group, even though they have different type (mountain fold v.s. valley fold). Since any of the real vertices can cover all the crease groups, the 3×3 Miura crease pattern has only one essential vertex which could be v_1 or v_2 or v_3 or v_4.

By gathering crease lines from a large crease pattern into crease groups, the DOF of the origami can be reduced from the number of crease lines to the number of crease groups. Moreover, by identifying essential vertices, we only need to check the local foldability (Eq. (3.1)) on *essential vertices*, a much smaller subset of real vertices than the number of all the real vertices. Table 3.3 reports the size of crease groups and essential vertices of 6 crease patterns. As we can also see in Table 3.3, using symmetry and essential vertex significantly reduces the computation time for finding a valid folding motion. We also tested the running with and without collision detection. From Table 3.3 we can see when we use full DOF for planning, the majority of the time is spent on finding valid configuration, collision detection takes only about 2% of the running time for folding the 5×5 Miura crease pattern. However, when we use symmetry property and essential vertex, the running time

reduced significantly, collision detection (with almost the same amount of computation) then dominates the running time which takes about 83% on average.

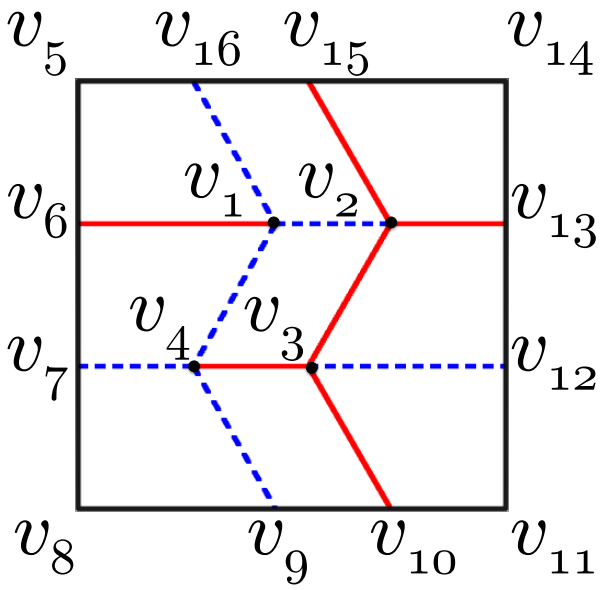
3.3.2 Reusing Folding Path

Given a crease pattern (tessellation), if this crease pattern is rigid foldable, it is expected that the folding angles of all crease lines in the same crease group remains identical even when planning is done using the full DOF.

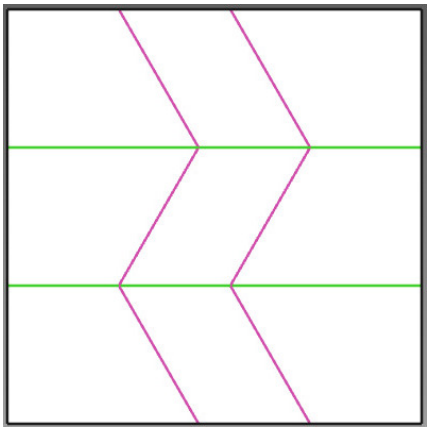
Model	RV/EV	SYM	EV	DOF	MI	Time (sec)	CD (%)
3×3 Miura	4/1	×	×	12	25	0.037	27.03
		○	×	2	5	0.016	56.25
		○	○	2	5	0.014	64.29
5×5 Miura	9/1	×	×	40	500	1.681	2.02
		○	×	2	5	0.098	81.63
		○	○	2	5	0.082	85.37
24×24 Miura	529/1	×	×	1680	N/A*	N/A*	N/A*
		○	×	2	100	35.278	78.32
		○	○	2	100	32.402	99.51
4×6 Waterbomb	15/3	×	×	50	5	0.040	77.50
		○	×	4	5	0.037	81.08
		○	○	4	5	0.034	88.24
8×10 Waterbomb	63/3	×	×	182	5	0.406	79.80
		○	×	4	5	0.332	90.36
		○	○	4	5	0.310	96.45
12×22 Waterbomb	231/3	×	×	758	5000	499.048	11.27
		○	×	4	5	3.893	91.75
		○	○	4	5	3.160	97.59

Note that the running time were obtained under 5% deformation upper bound. RV=Real Vertex, EV=Essential Vertex, SYM=Symmetry, MI=Maximum Iteration, CD=Time cost for Collision Detection. The symbols ○ and ×, in the columns of SYM and EV, indicate if symmetry and essential vertex are used or not. *The planner failed to find a valid path within the time limit due to high DOF.

Table 3.3: Path Planning Time using Symmetry.

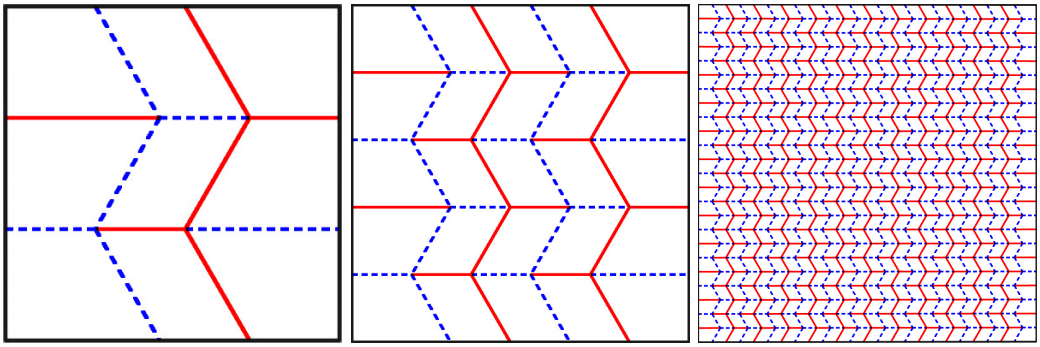


(a) Crease pattern

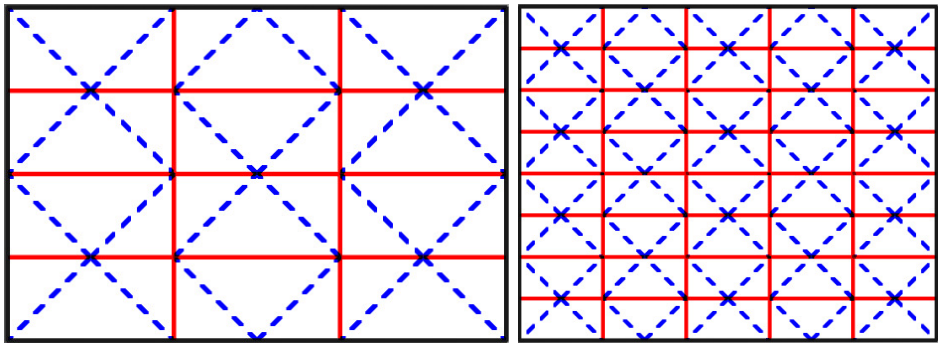


(b) Crease groups

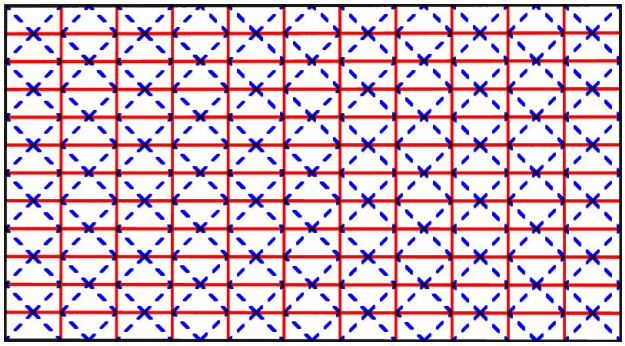
Figure 3.8: Crease groups of a 3×3 Miura crease pattern. Crease lines belong to the same crease group are shown in the same color.



(a) 3×3 Miura (b) 5×5 Miura (c) 24×24 Miura

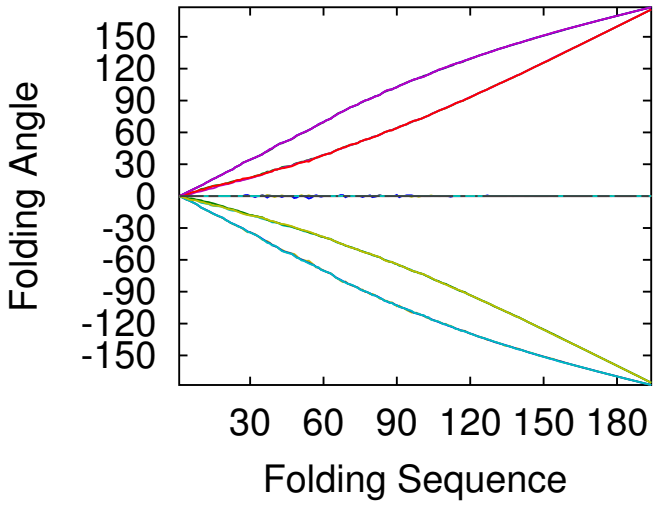


(d) 4×6 Waterbomb (e) 8×10 Waterbomb

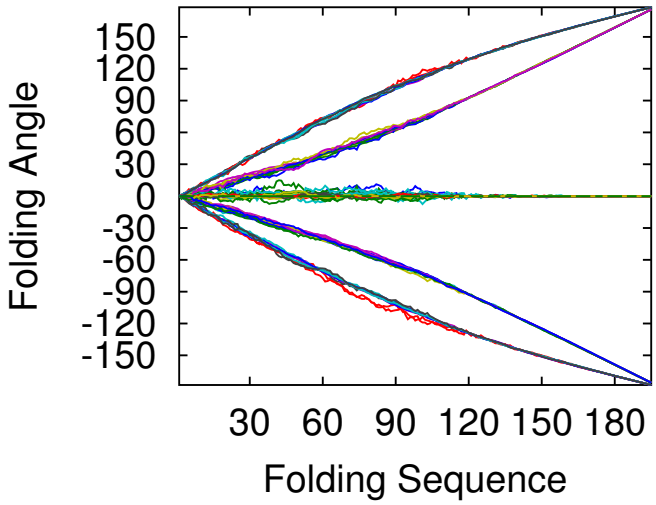


(f) 12×22 Waterbomb

Figure 3.9: Origami tessellations used in our experiments.



(a) 3×3 Miura DOF=12



(b) 5×5 Miura DOF=40

Figure 3.10: Folding paths found without using symmetry information.

Chapter 4: Polyhedra Unfolding

4.1 EU: A Genetic Algorithm for Unfolding

As mentioned in Chapter 2, the unfolding of a Polyhedron can be obtained by finding a spanning tree of the dual graph of the mesh. Heuristic methods assign different weights on the dual edges such that the minimal spanning tree of the dual graph has a high probability to be a net. In this section, we propose EU (Evolution of Unfoldings), a genetic algorithm to evolve the unfoldings to nets.

4.1.1 Genetic Representation

Inspired by the heuristic methods for unfolding, we use the dual edge weights to represent the gene. The length of the gene is equal to the number of the non-boundary edges of the mesh and the range of the weight is from 0 to 1.

4.1.2 Fitness Evaluation

We evaluate the unfolding using the fitness score defined as: $f = -(\lambda_o N_o + \lambda_l N_l)$, where N_o is the number of overlaps in the unfolding and N_l is the number of hyperbolic vertices that cause local overlaps in the unfolding, λ_o and λ_l are their coefficients. Since local overlaps are harder to resolve than global ones thus they play a more important role in the fitness function. Once the fitness score becomes zero, we found a net.

4.1.3 Population Generation

We use 3 existing heuristic methods: $Steepest-Edge$, $Flat-Tree$, $Minimum-Perimeter$ [3] to generate the initial population of p individuals which give us a good initialization with a large variance in the unfoldings.

4.1.4 Selection, Mutation and Crossover

Tournament selection, uniform crossover and Gaussian mutation are used to create and mutate individuals. In each generation, k children will be created to replace the worst k parents.

4.1.5 Experimental Results

Without notice, we use the following parameters for all our experiments. Initial population $p = 400$, tournament size $t = 7$, new children each generation $k = 50$, crossover probability $= 0.4$, mutation probability $= 0.02$, mutation variance $= 0.3$, maximum generation $g_{max} = 5000$. We use $\lambda_o = 1, \lambda_l = 10$ for the weights.

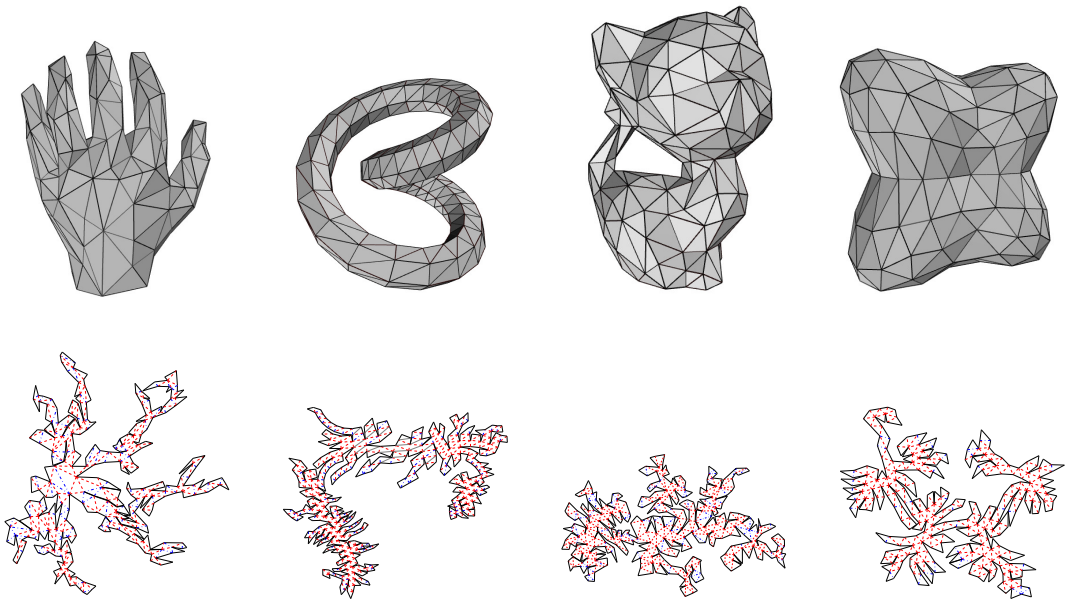
The nets found by the proposed method are shown in Fig. 4.1. We show the average and best fitnesses of each generation in Fig. 4.2 for the Bunny model and MoneyBox model (shown in Fig. 4.1(c)), from which we can see that the proposed method quickly converges to a valid solution after 200 generations.

4.2 Simultaneously Segment and Unfold

Segmentation and unfolding are both edge-cutting operations that determine the foldability of a mesh, thus should not be decoupled. In this section, we will discuss a method that first estimates the likelihood of every pair of faces that can be unfolded together without overlapping, and then segments the mesh into face clusters that have high probabilities of becoming valid nets. Upon the failure of unfolding a cluster in the segmentation, the cluster is further segmented using the learned likelihood until all clusters are unfolded. Fig. 6.2 provides an overview of the proposed method.

4.2.1 Learn from Failed Unfoldings

To determine the likelihood of every pair of faces that can be unfolded together without overlapping, the proposed method unfold the mesh multiple times using existing heuristics,



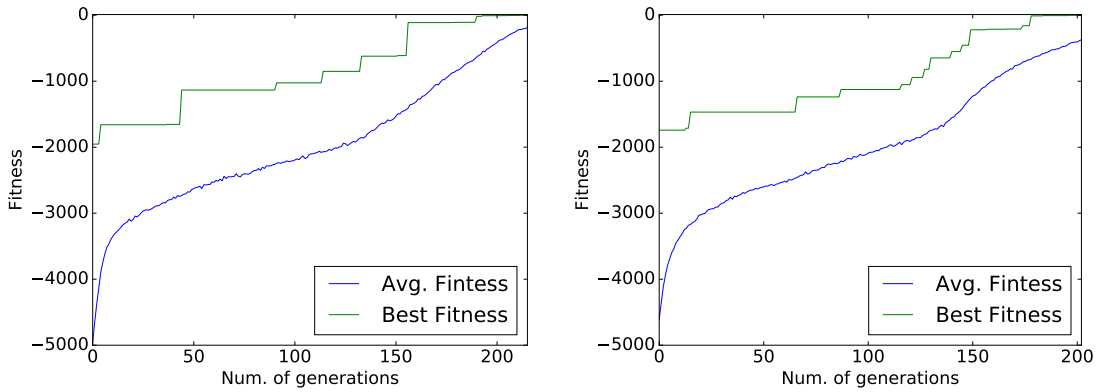
(a) Hand (b) Knot (c) MoneyBox (d) Pillow

Figure 4.1: Meshes and its nets found by the proposed method.

such as steepest edge for example. After the mesh is unfolded m times using these unfolding heuristics, the proposed method analyzes whether two faces f_1 and f_2 belong to the same connected component without overlapping in all of these unfoldings. If f_1 and f_2 belong to the same non-overlapping connected component n times among m trials, then their foldability likelihood is simply n/m. Thus, the result of this learning step is a symmetric similarity matrix, called "foldability matrix", in which a large value means that two faces are likely to be unfolded without overlapping. We next describe each of these steps in detail.

4.2.2 Unfold the Mesh Multiple Times

As discussed earlier, an unfolding of a mesh is closely related to the minimum spanning tree of the dual graph of the mesh. That is, a set of edge weights determines a specific unfolding. Therefore, m unfoldings can be obtained by m sets of edge weights. If the random unfolding heuristic is used, then m unfoldings can simply be created by drawing $m||E||$ arbitrary numbers. If steepest edge or flat tree heuristics are used, then we draw



(a) Bunny-348 (b) MoneyBox-396

Figure 4.2: Average and best fitness of each generation.

m random unit vectors and use these vectors to determine the edge weights. Alternatively, instead of drawing random vectors, we also experimented with the surface vectors of the mesh, such as outward normal vectors of faces and vertices and vectors parallel to the edges. We found no differences of how the vectors are selected. The number of vectors plays a more influential factor.

4.2.3 Analyze an Unfolding

An edge unfolding usually contains multiple overlaps and is not a valid net. However, we can still obtain valuable information about the foldability of the mesh from an invalid net.

Given an edge unfolding represented by a tree $\mathcal{T}$, and let $\mathcal{L} = \{(f_i, f_{j\neq i})\}$ be a list of overlapping face pairs (f_i, f_j) in $\mathcal{T}$. If $\mathcal{L}$ is empty, then we found a valid net; otherwise we will use $\mathcal{L}$ to determine the foldability likelihood of $\mathcal{T}$. Given a face f of $\mathcal{T}$, let $CC(f)$ be a set of non-overlapping faces that contains f. We say that $CC(f)$ is *maximized* if no additional faces can be added to $CC(f)$ without overlapping with the members of $CC(f)$. To compute the maximized $CC(f)$, we start with $CC(f) = \{f\}$, and then iteratively test the faces of $\mathcal{T}$ adjacent to the current $CC(f)$ in breadth-first search manner, i.e., only expand $CC(f)$ via the fold edges and not the cut edges. Let f' be an adjacent facet that

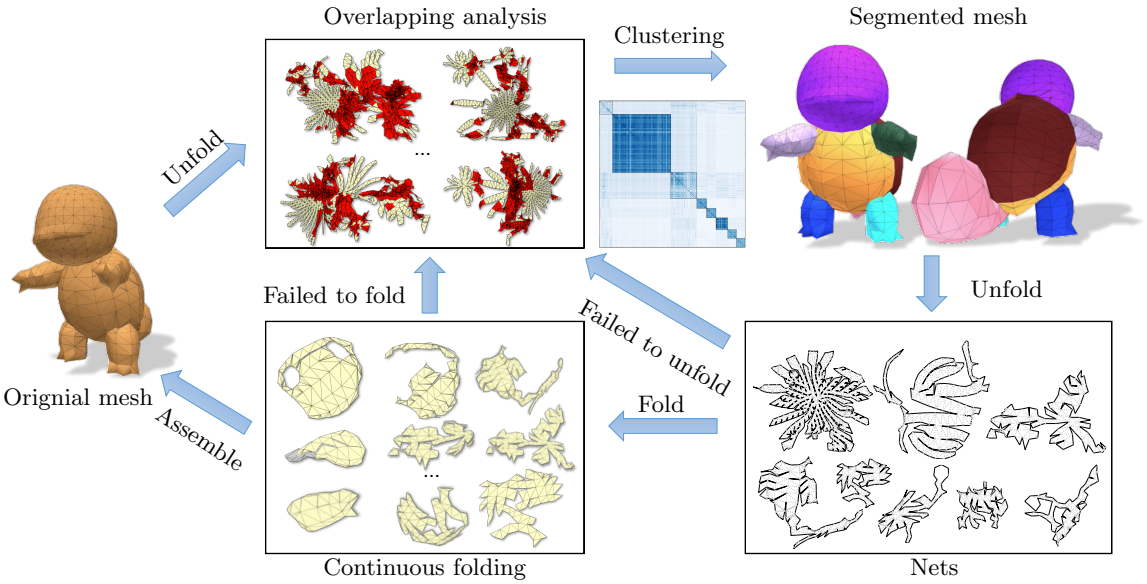


Figure 4.3: Overview of the proposed method. Overlapped faces are shown in red in the 'Overlapping analysis' box. Foldability matrix after clustering is shown below the 'Clustering' arrow, in which rows are sorted by cluster id. Pixel $p(i, j)$ indicates the probability that face f_i does not overlap with face f_j in the unfoldings, the darker the higher. Each block along the diagonal represents one cluster. If any of segment was failed to unfold to net or failed to fold back, we can further segment it using the proposed method. This process is repeated until all segments have nets and can be continuously folded back to 3D.

does not overlap with the facets of $CC(f)$, then $CC(f) = CC(f) \cup f'$.

After the maximized $CC(f)$ is found, the foldability matrix $\mathcal{M}$ is updated so that all elements between f and $f' \in CC(f)$ are increased by one, i.e. $\mathcal{M}(f, f') = \mathcal{M}(f', f) = \mathcal{M}(f, f') + 1, \forall f' \in CC(f)$. This operation is repeated for all faces for a given $\mathcal{T}$.

When multiple unfoldings are performed on the same mesh, the foldability likelihood matrix $\mathcal{M}$ accumulates the likelihood estimation.

4.2.4 Segment

After the foldability likelihood of all pairs of faces is determined, we use spectral clustering to cluster the faces. Although other clustering methods can also be used, we found that spectral clustering gives consistent and better results. For example, we attempted to use Lloyd's algorithm that ensures all clusters are made of connected faces, but this approach usually results in sub-optimal clusters because the idea of cluster center cannot be easily defined. An example of the clustered foldability matrix using spectral clustering can be found in Fig. 4.4.

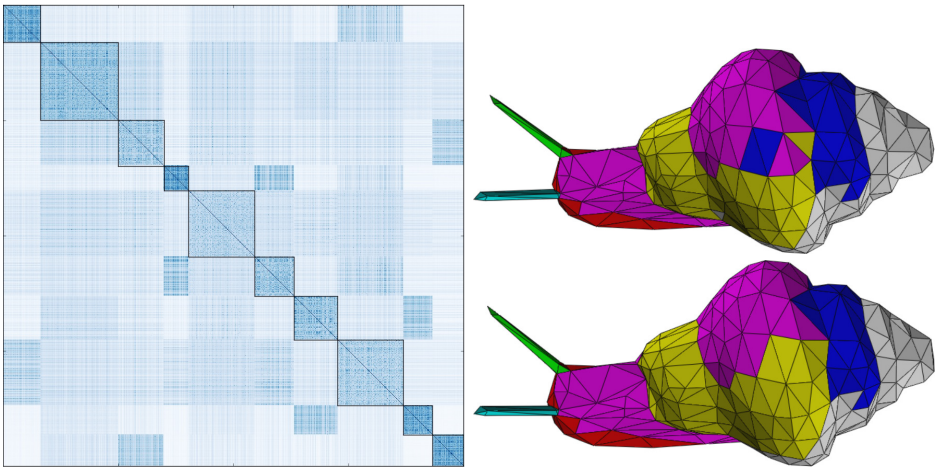


Figure 4.4: Left: Clustered foldability matrix of the monkey model. A darker pixel indicates higher possibility of unfolding the corresponding face pair without overlapping. 10 blocks along the diagonal represent the 10 clusters found and correspond to the segmentation of the monkey model. Right: Before (top) and after (bottom) isolated facets are reassigned.

Spectral clustering does not consider face adjacency, therefore some triangles may be separated from the main components. Our experiments show that, if there exist multiple connected components in a given cluster, these isolated components are much smaller in size than the main component in the given cluster. Therefore, a post-processing step is suffice to enforce every cluster contains only connected facets. Given that f is a face disconnected from the the largest component in its cluster C_i and is adjacent to the clusters $\{C_{j\neq i}\}$. Then f is reassigned to a new cluster C_j such that

$$\arg\max_j \sum_{f' \in C_j} \text{area}(f')\mathcal{M}(f, f') \ .$$

The snail model shown in Fig. 4.4 illustrates an example before and after the isolated components are reassigned.

4.2.5 Results

We show that the nets produced by the proposed method can be used to create more complex paper craft (e.g., 3000 faces in 36 hours in Fig. 4.5) comparing to the results reported in the literature (e.g., 347 faces in 25 hours [16]).

4.3 Continuous Unfolding of Polyhedra

Once we obtained a net for the mesh, the next question we want to address is that whether there exists a continuous folding motion that transforms the net back to the original mesh. Since the net has a very high degree of freedom which equals to the $|F| - 1$ where $|F|$ is the number of faces, which could be hundreds or even thousands that makes traditional motion planners failed to work. We employ the method from [11] to plan continuous folding motion for the net. Instead of sampling in the continuous domain that has extremely low probability (close to 0) to generate a valid configuration, this method only samples in discrete domain, the folding angle of each crease line is sampled from $\{0, \rho_i\}$, where ρ_i is the target folding

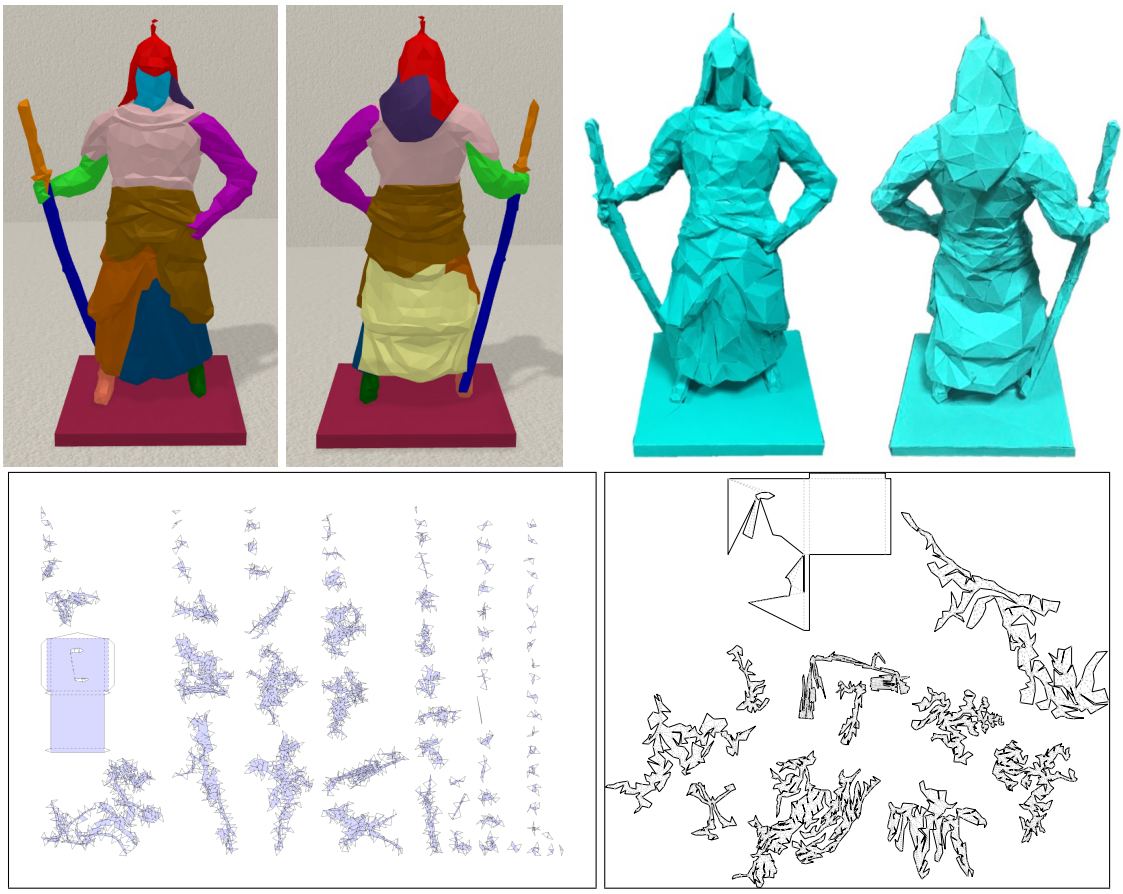


Figure 4.5: **Top left:** The statue of Korean general Yi Sun-sin (3000 triangles) is decomposed into 11 clusters by the proposed method. **Top right:** The paper craft of the model, which is 9.7 cm wide, 9.7 cm deep and 21.2 cm tall. **Bottom left:** The nets generated by Pepakura (79 parts). **Bottom right:** The nets generated by the proposed method (11 parts). The unfolder developed by [2] is unable to unfold the model perhaps due to that the mesh is not water tight.

angle of the i-th crease line. There are $2^{|F-1|}$ possible states in the configuration space. Fig. 4.6 shows the valid configuration ratios defined as $|C_{valid}|/|C_{sampled}|$ of various models and its nearly convex decomposition components whose DOFs range from 7 to 479 for two sampling strategies, from which we can see that sampling in discrete domain maintains a much higher probability, which is exponentially higher than uniform sampling.

Also, we can see that for discrete domain sampling, nearly convex decomposed patches have a higher valid configuration ratio compare to that of original mesh which implies they

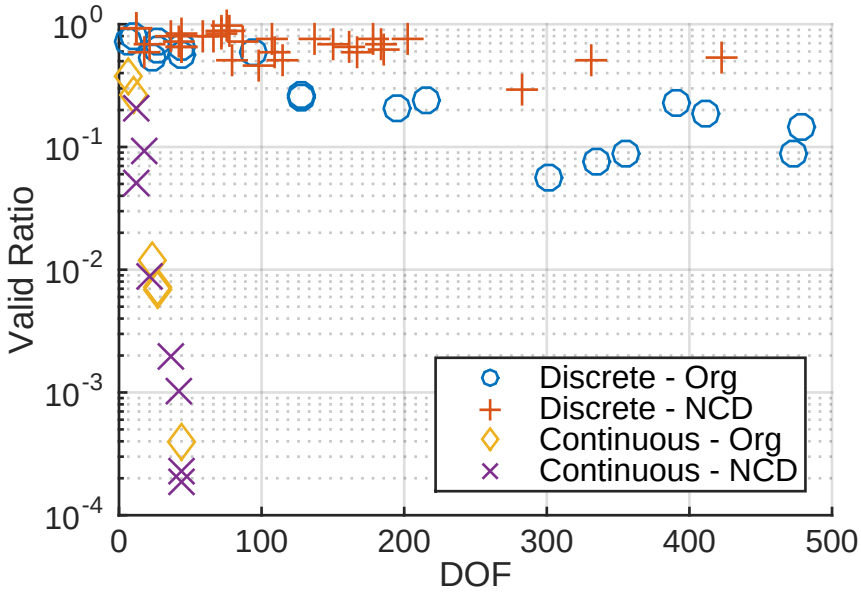


Figure 4.6: Valid configuration ratios of various original models (Org) and their components decomposed by nearly convex decomposition (NCD) under two sampling strategies.

are easier to fold. With those valid configurations sampled from discrete domain, Lazy-PRM ([37]) is employed to find feasible folding paths. Thanks to nearly convex decomposition, those decomposed patches have a much higher probability that the folding path is a straight line in the high dimensional configuration space and the feasibility of the path can be tested in $O(|F|^2)$ with a naive collision detection method. The folding process of bunny mesh found by the motion planner using discrete domain sampling is shown in Fig. 4.7. We encourage readers to visit our web-based interactive folding process visualizer at `http://masc.cs.gmu.edu/origami/folder.html` to experience the complexity and beauty of continuous folding.

4.4 Polyhedra Fabrication via Mesh Convexification

Paper crafting enables us to fabricate a target surface from one or multiple sheets of papers which are easier to manufacture and transport [12, 13, 17]. This technique can be used to

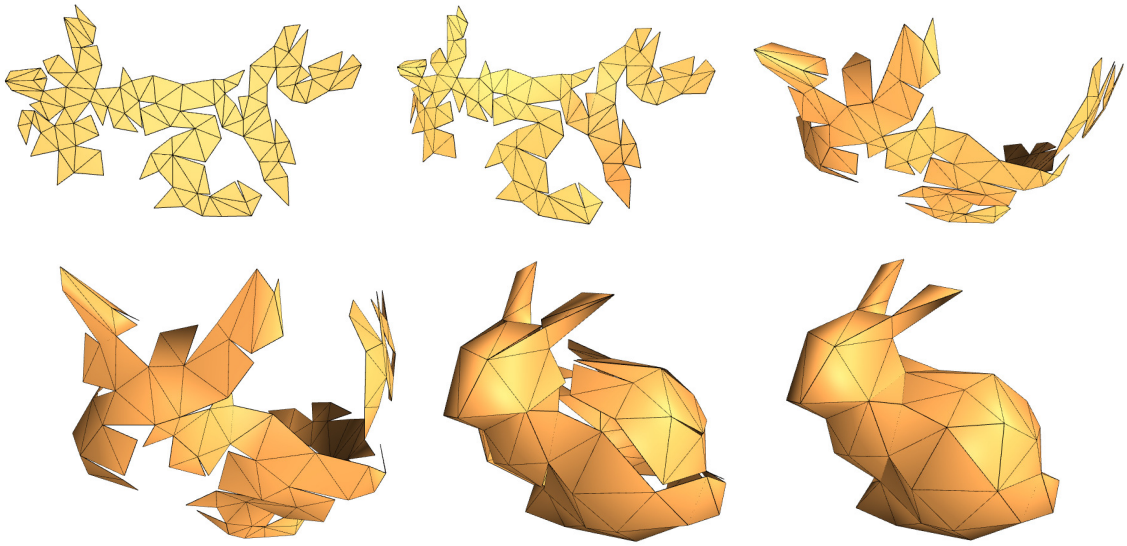


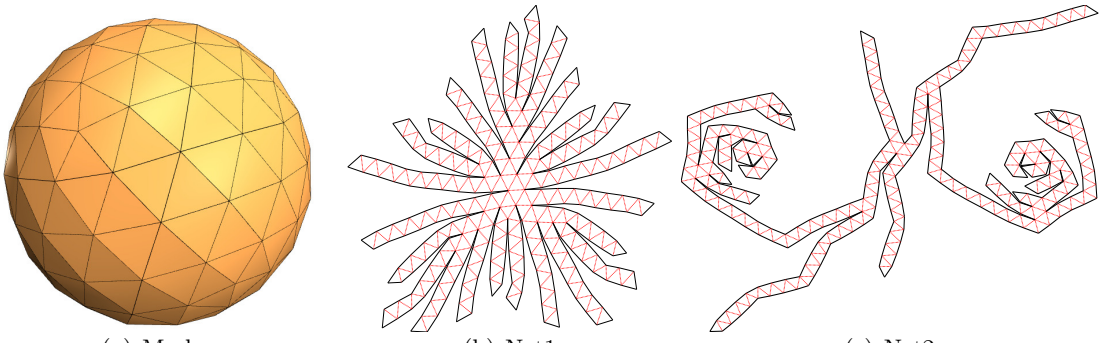
Figure 4.7: Continuous folding process of the Bunny model.

design self-folding robot [4] with rigid materials that can fold itself from a flat sheet to a 3D functional shape via uniform heating [39], magnetism [20] or lighting [40]. Designing a paper craft usually involves two main foldability analysis steps. The first step is to find a 2D representation (which could be a net, a crease pattern or a developable surface) whose folded shape approximates entire or part of the mesh. We call this an *instantaneous unfolding* problem, since the solution will be a function that instantaneously transforms the polyhedron to the 2D representation without going through any intermediate configurations. In this dissertation we focus on the *net* representation, which is the unfolding of the polyhedron that does not contain overlaps. Finding a valid net of a given polyhedron is known to be nontrivial because a polyhedron with $|F|$ faces can have approximately $2^{\sqrt{|F|}}$ different unfoldings and most of them contain overlaps especially for non-convex polyhedra. Suppose that we obtained a net, the second step is to find a foldable path that transforms the net to its folded shape continuously without self-intersection. We call this a *continuous folding* problem. Unfortunately, none of previous paper crafting works take continuous folding into consideration and assume that the net is always foldable. However, this assumption

is not always correct especially when folding non-convex patches with rigid materials. In this dissertation, we model the net as a multi-link articulated robot and solve the folding problem using motion planning approach. The dimensionality of the configuration space is equal to $|F| - 1$, where $|F|$ is the number of faces in the net, and it could be hundreds or even thousands.

Though the problem of finding edge unfolding for convex polyhedra is still open, heuristic methods work well in practice. Most, if not all, nets of convex polyhedra can be obtained in $O(|F|log|F|)$. Furthermore, the start state, i.e., the flat net and the goal state, i.e. original *convex* polyhedron, may sometimes be linearly connected in the configuration space. This property significantly reduces the path planning time to find a continuous folding motion. Fig. 5.3 shows a convex polyhedron and its nets, one of whose folding path is a straight line in the configuration space. However, when dealing with non-convex shapes, previous works [2, 11] show that each step itself is challenging. Thus we employ mesh inflation to remove local concave features and nearly convex decomposition method to segment a mesh into several part-aware and nearly convex patches. Then both instantaneous unfolding and continuous folding problems are solved for each patch separately. However, since the patch is not exact convex, no heuristic methods guarantee to find a net for it (and there exist some non-convex polyhedra that can not be unfolded), thus we employ a genetic algorithm to find nets for those nearly convex patches. Once we obtained the net, motion planning algorithm is used to find a continuous folding path for it. Nearly convex decomposition shows great advantages in both steps. After we found the feasible folding paths for all the patches, the only left step is to assemble the folded shapes. Our experimental results show that the mesh convexification makes each step several folds faster in terms of total running time than working on the original mesh alone. It also makes manufacturing and assembling easier.

Given a mesh $\mathcal{M}$, we first inflate the mesh to reduce surface concavity (Section 4.4.1), then remove structural concavity by decomposing the mesh into several part-aware, nearly convex components (Section 4.4.2). For each component we find a net using a genetic-based



(a) Mesh (b) Net1 (c) Net2

Figure 4.8: A convex mesh and its nets found by two heuristic methods: Steepest Edge for net1 and Flat Tree for net2. Both nets were obtained within 0.01s. Net1's folding path is a straight line in the configuration space (only 1 edge was checked). For net2, in order to find a feasible path, 1056 edges need to be checked (approximately 1000 times slower) on average.

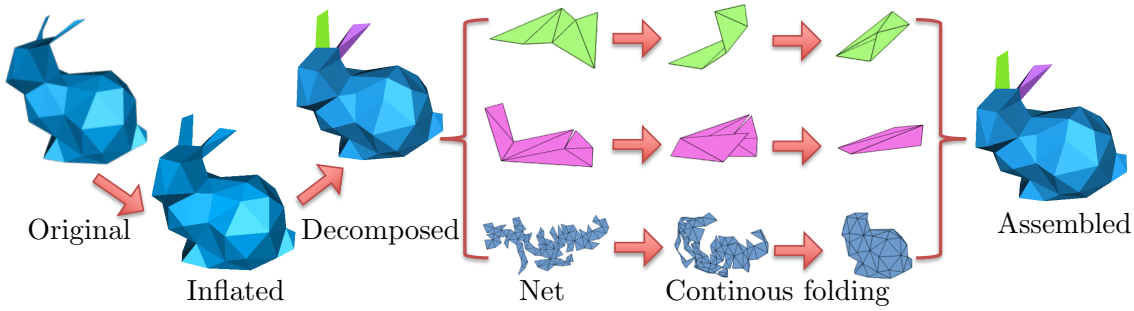


Figure 4.9: Pipeline of the fabrication of the model.

algorithm (GA) [19], the initial population are generated using heuristic methods. Once we obtained the net, motion planning is introduced to find a feasible path that folds the net back to its 3D shape continuously to ensure we can build a physical copy even use rigid materials instead of flexible materials, such as paper which could be easily bent during folding. Finally, all the components can be assembled. The pipeline of proposed approach is shown in Fig. 4.9.

4.4.1 Reduce Local Concavity via Mesh Inflation

Hyperbolic vertices are the main sources that cause existing unfolding methods fail to find a net. Because every hyperbolic vertex must be incident to at least two cuts, reducing the number of hyperbolic vertices implies the reduction in the variance of the number of cuts of each vertex and therefore simplifies the unfolding. Many of these hyperbolic vertices can be removed without affecting the overall shape. Inspired by physically based simulation to inflate a concave mesh into a balloon, We propose to use the idea to pop up small dents on the mesh and reduce the number of hyperbolic vertices so that the computation time of finding a net can be reduced. In particular, we will show that the net of an inflated model can be created with few modifications from an invalid unfolding generated by heuristic methods, which are usually designed for convex shapes.

Uniform (Unconstrained) Inflation

Force caused by air pressure should be uniformly distributed on the entire face, in this dissertation, we simplify the model and assume forces only exist on the local region of each vertex v. Then the force on the vertex is a weighted average of forces on adjacent faces,

$$\vec{f_p} = \lambda_p p \sum_i \phi_i \vec{n}_i \tag{4.1}$$

where λ_p is a coefficient, p is the pressure, ϕ_i is the section angle of each adjacent face and $\vec{n}_i$ is the normal direction of that face. Since $\vec{f} = \vec{p}A = \vec{n}pA$, at the local disk region around the vertex v, ϕ is proportional to A, thus we can use ϕ as the weight to compute the force contributed by each adjacent face.

During inflation, vertices moved, edges deformed. By assuming the mesh is made of elastic materials, the force applied to the vertex v_i due to stress on edge $\overrightarrow{v_i v_j}$ can be defined as,

$$\vec{f_{ij}} = \frac{E_e \Delta L \overrightarrow{v_i v_j}}{||\overrightarrow{v_i v_j}||} \ , \tag{4.2}$$

where E_e is the elastic modulus of the material, ΔL is the length changed of the edge defined as $\Delta L = L_{cur} - L_{org}$. Since we would like to have the mesh inflated, only stretched case ($\Delta L > 0$) is considered, otherwise $\vec{f_{ij}} = 0$.

The total elastic force on vertex v_i is the summation of forces on adjacent edges,

$$\vec{f_e} = \sum_j \vec{f_{ij}} \ . \tag{4.3}$$

The total force on each vertex is then,

$$\vec{f} = \vec{f_p} + \vec{f_e} \ . \tag{4.4}$$

The position $\vec{P}$ of the vertex is updated in an iterative manner and we assume each vertex has the same unit mass and zero mass elsewhere,

$$\vec{P}_{t+1} = \vec{P}_t + \Delta t \vec{f} \ . \tag{4.5}$$

Constrained Inflation

Although uniform inflation works well in practice in terms of reducing the number of hyperbolic vertices, it introduces several undesired properties for our application: convex region will be inflated as well and flat surfaces do not maintain.

Convex and Concave Edges In order to maintain convex regions and keep deep concave regions, we multiply a stiffness ratio to the edges based on their current folding angle,

$$\vec{f_e'} = \begin{cases} \lambda_{se}\vec{f_e}, & \text{edge is convex or deep concave} \\ \vec{f_e}, & \text{otherwise} \end{cases} \tag{4.6}$$

where λ_{se} is the stiffness ratio coefficient.

Virtual Edges We add virtual elastic edges between the current position of the vertex P' and its original position P to penalize large displacement of the elliptic vertex. The pulling force applied to the vertex is similar to the one shown in Eq. (4.2),

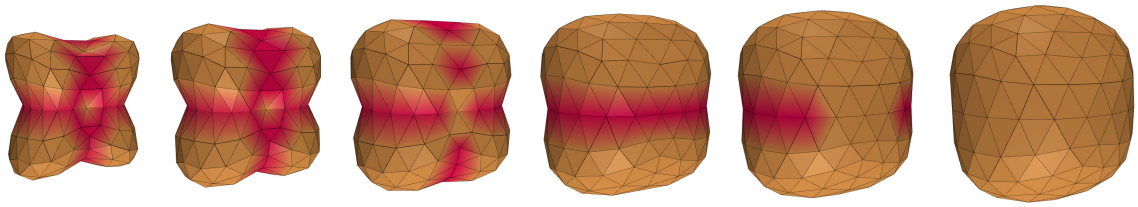
$$\vec{f_v} = \frac{\lambda_v E_v \Delta L \overrightarrow{PP'}}{||\overrightarrow{PP'}||} \tag{4.7}$$

$$\lambda_v = \begin{cases} 1, & \text{hyperbolic vertex} \\ \lambda_{sv}, & \text{elliptic vertex} \end{cases} \tag{4.8}$$

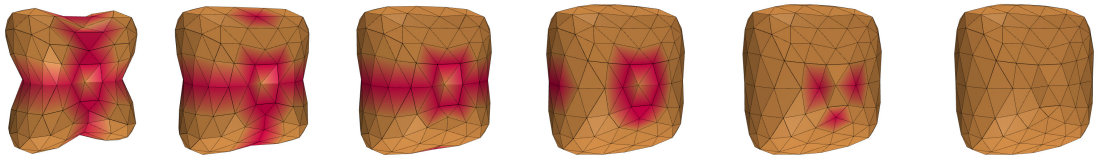
where λ_{sv} is another stiffness ratio coefficient.

Finally, Eq. (4.4) can be rewritten as,

$$\vec{f} = \vec{f_p} + \vec{f_e'} + \vec{f_v} \tag{4.9}$$



(a) 0.0000(44) (b) 0.0918(48) (c) 0.1726(36) (d) 0.2371(20) (e) 0.2681(14) (f) 0.3182(0)



(g) 0.0000(44) (h) 0.0539(38) (i) 0.0705(28) (j) 0.0807(20) (k) 0.0830(6) (l) 0.0839(0)

Figure 4.10: Inflated meshes with different inflation rate (IF). IF is shown as the caption, the number in the parentheses indicates the number of hyperbolic vertices. **Top:** uniform inflation, **Bottom:** constrained inflation. Hyperbolic vertices are shown in red.

Implementation

Mesh inflation methods are implemented in JavaScript with an interactive UI which is available at `http://masc.cs.gmu.edu/origami/inflation.html`.

Experimental Results

An example of mesh inflation can be found in Fig. 4.10, from which we can see that inflation can reduce the number of hyperbolic vertices effectively. However without any constraints, the mesh will be inflated to a sphere-like shape finally (if the pressure is high enough). Though all the concave features were removed, the volume of the mesh increased radically. Constrained inflation helps to achieve similar results with much less inflation. We also measure the unfolding time, folding time and folding success rate on original mesh and inflated ones. From Fig. 4.10 we can see that inflation removes hyperbolic vertices which helps GA to find the net more efficiently and more effectively. It also helps to find the feasible folding path more efficiently and more effectively. The reported unfolding time is the average of 30 runs on each inflated mesh; path planning time is the average of 20 runs on each found net; time limit for each trail is 600 seconds. And we can see that constrained inflation achieves similar or better results with much less deformation compared to uniform inflation.

4.4.2 Reduce Concavity via Decomposition

Shape segmentation is widely used in paper crafting [12, 13]. To increase the chance of finding a net/feasible folding path for that net, several desired properties should be provided in the final decomposition: (1) small in size, (2) convex or at least nearly convex, (3) part aware, (4) and disjoint, if possible.

Small in size. One arguable question is how many components should one generate for the mesh. If the answer is one, then we stay with the original mesh, both unfolding and folding are challenging problems which might be even unsolvable (non-convex polyhedra do not always have a net). If the answer is $|F|$, the number of faces, then both folding and

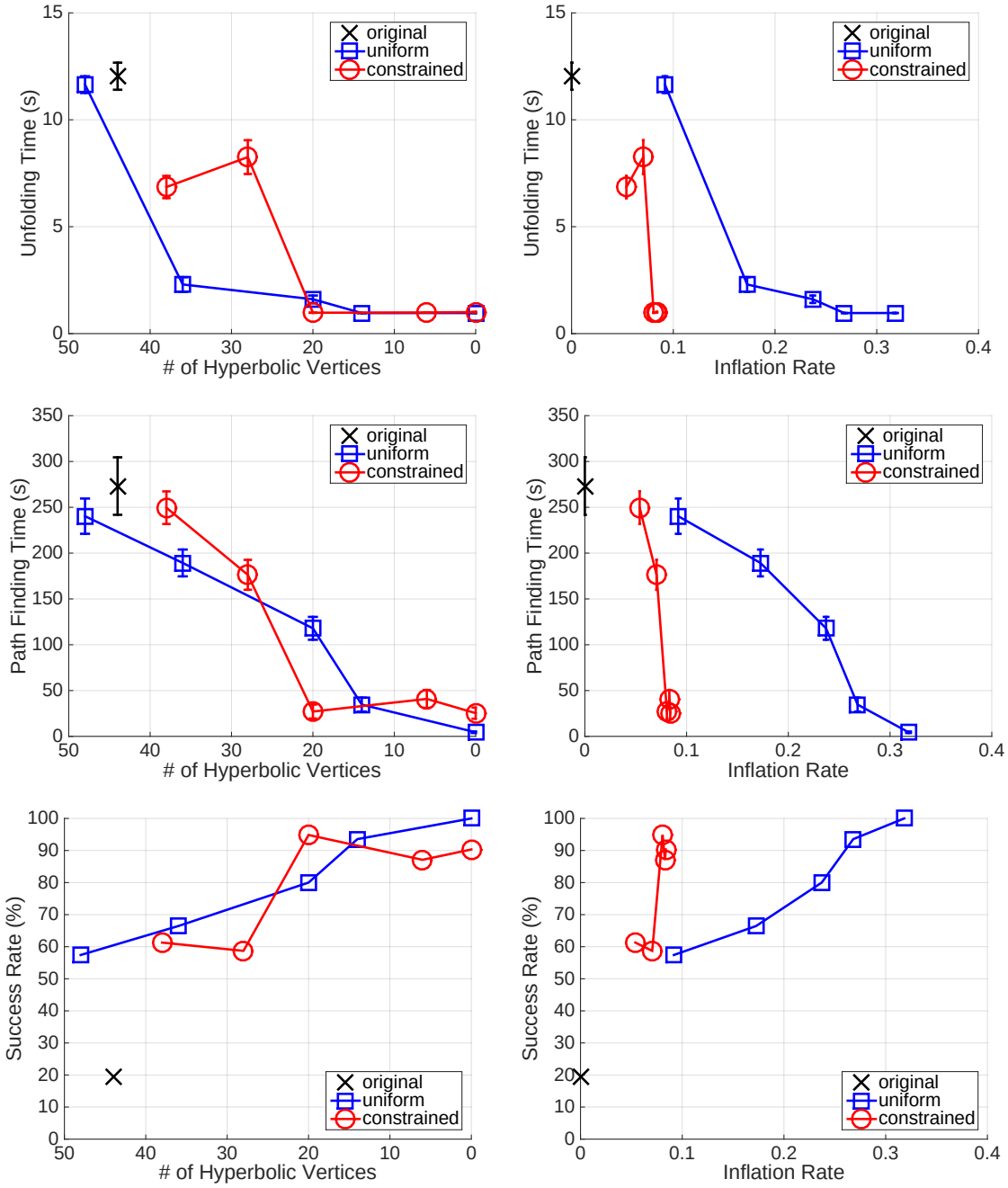
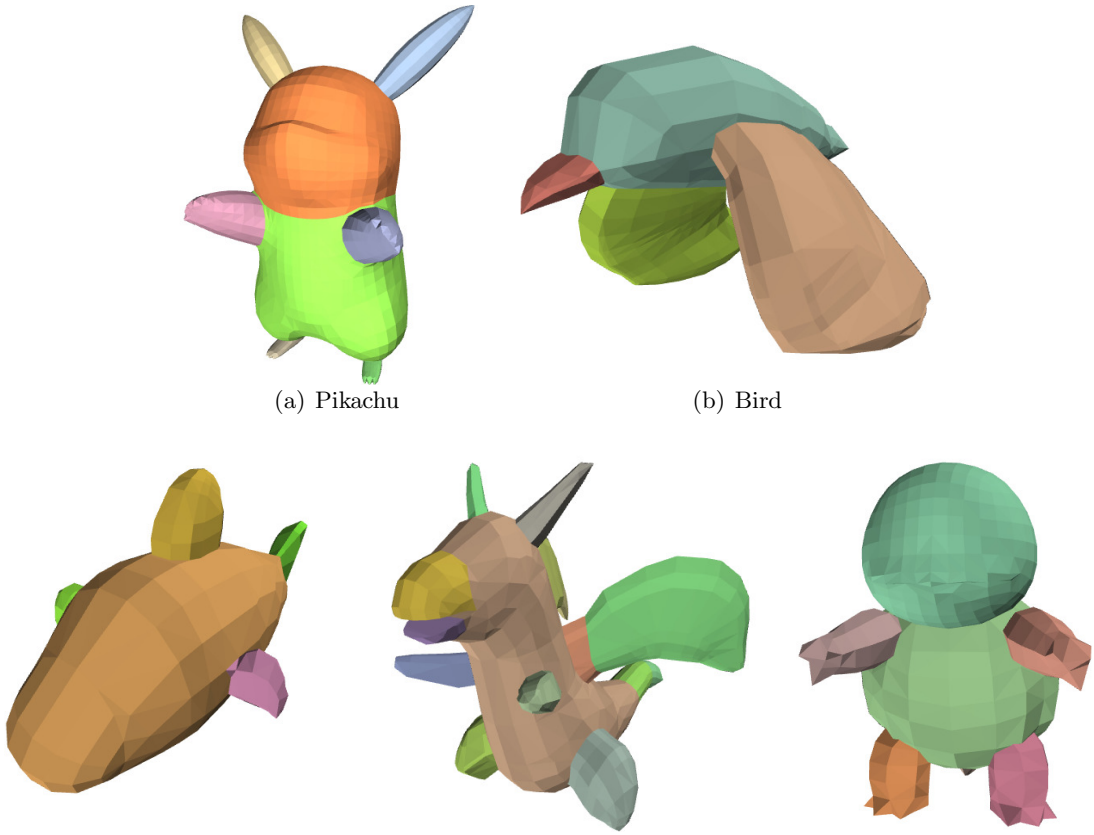


Figure 4.11: **Left**: Unfolding time, path finding time, path finding success rate as functions of number of hyperbolic vertices. Note: x axis is in reverse order. **Right**: that as functions of inflation rate.



(a) Pikachu (b) Bird

(c) Fish (d) Dragon (e) Squirtle

Figure 4.12: Part-aware nearly convex decomposition. All components contain concavity smaller than 0.05.

unfolding can be done in $O(1)$, however, it makes manufacturing and assembling impractical. Fortunately, the number of decomposed patches can be controlled by user parameters.

Convexity. Convex objects are much easier to fold and unfold. Exact convex decomposition produces overwhelmingly many components. Segmentation methods produce nearly convex components produces decomposition with reasonable size. However, if the component is not convex which means it is not guaranteed to have a net, and might be difficult to find a net and plan folding motion.

Component semantic. Part-aware decomposition not only makes both unfolding and folding much easier but also provides a natural experience for manufacturing and assembling.

Disjoint components. Replacing each nearly convex patch with its convex hull to

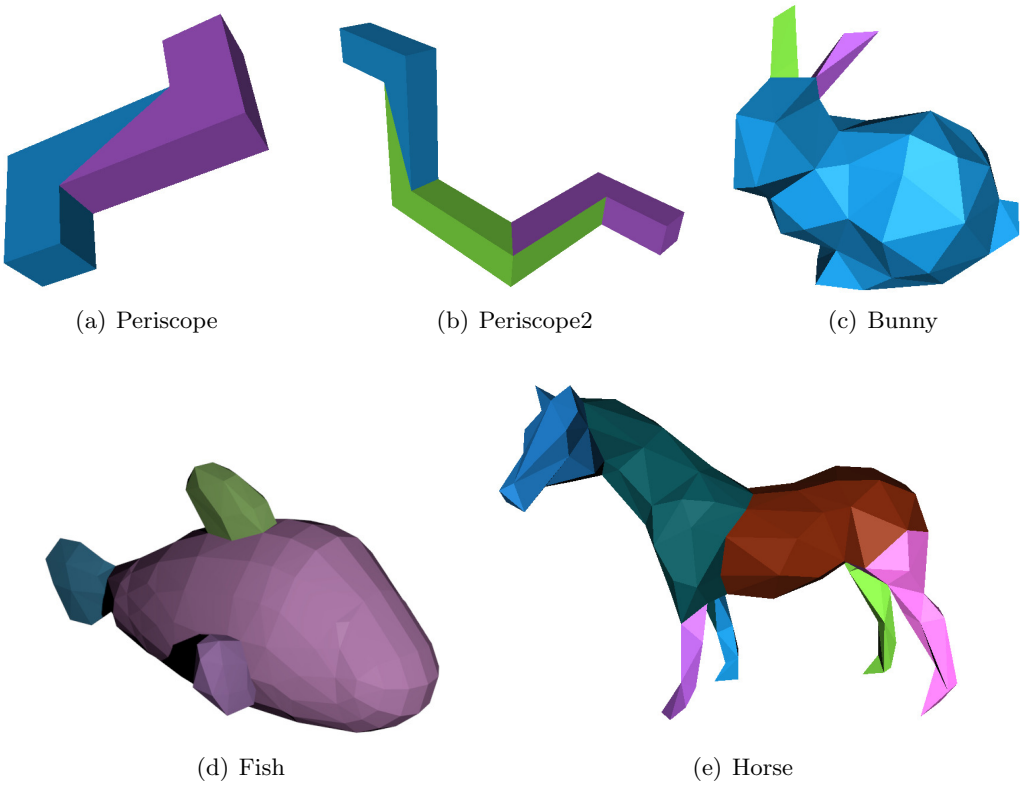
approximate the original mesh might be a key to both unfolding and folding problems. Unfortunately, convex hulls of decomposed components usually collide with each other which makes assembling impossible. How to segment the polyhedron that the convex hulls of the decomposed components do not collide with each other is still an open problem.

In this disseration we employ a method called Convex Ridge Separation (CoRiSe) [34] to decompose a 3D mesh to part-aware with controllable concavity of all components in the decomposition. Examples of five decompositions are shown in Fig. 5.4. All components in these examples contain concavity smaller than 0.05. The concavity of a shape is defined as the maximum distance between the convex hull and the shape. It is important to note that other segmentation methods can also be incorporated with the proposed framework as long as the convexity of the component can be bounded. For example, an alternative approach can use Continuous Visibility Feature [41] to repetitively segment the shape until certain desired convexity is reached.

Experimental Results

Experiment Setup We implemented the proposed unfolding/folding methods in C++, which will be released to the public domain. All data reported here were collected on a 2012 MacBook Pro with a 2.9 GHz Intel Core i7 CPU and 16GB Memory running Mac OS X. Meshes used in the experiment are shown in Fig. 5.17 as their decomposed states, components are shown in different colors.

Running Time We compare the running time of unfolding the entire mesh and sum of running time of unfolding the decomposed components. The results (average running time of 20 runs) are shown in Table 4.2 from which we can see that NCD can significantly reduce the total running time especially on path planning, and the extra running time introduced by CoRiSe is negligible. For path planning, the main reason that NCD helps is that a convex shape usually has a good net that the start and goal are directly linear connectable [11, 42]. Thus we do not even need to plan the path, but only need to validate it. And as we can expect, NCD should have similar properties which will make continuous folding much



(a) Periscope (b) Periscope2 (c) Bunny

(d) Fish (e) Horse

Figure 4.13: Decomposed meshes.

easier. For the decomposition results of Periscope and Periscope2, all of their decomposed components have a linear path that directly connects start and goal, which significantly reduced the total running time (see Table 4.2).

For the fish model shown in Fig. 4.13(d), after nearly convex decomposition, the largest component (the body) still has 333 faces (original mesh has 474 faces). However, since all the components are nearly convex, both total running times (the summation of all components) reduced significantly especially for finding the continuous motion, which is 17.47x faster.

For the horse model shown in Fig. 4.13(e), we obtained the net for the original mesh which takes 219.87 seconds, however, we were not able to find a feasible folding path for the net in a certain amount of time (>1 hour). Using the proposed idea, we decompose the horse mesh to 7 part-aware and nearly convex components, in which, 4 for the legs, 1

Patch	DOF	LC	Finding Net	Path Planning
0	17	✓	0.00	0.01
1	17	✓	0.00	0.02
2	21	×	0.00	2.69
3	42	×	0.56	1.39
4	37	×	0.02	0.73
5	86	✓	4.84	0.31
6	75	×	0.58	1.60
Sub Total			6.00	6.75
Total			12.75	

LC: Whether C_s and C_g are linear connectable in C_{free}.

Table 4.1: Running time of the Horse model in seconds.

Model	#	DOF	CoRiSe	Net	Path	Total	Speedup
Periscope	1	27	-	0.08	7.05	7.13	59.33
	2	13 / 13	0.00	0.00	0.12	0.12	
Periscope2	1	43	-	0.21	12.40	12.61	48.50
	3	13 / 15 / 13	0.00	0.09	0.17	0.26	
Bunny	1	127	-	8.09	482.57	490.66	144.74
	3	4 / 6 / 115	0.02	2.90	0.47	3.39	
Fish	1	473	-	69.66	1848.86	1918.52	16.12
	5	26 / 34 / 36 / 44 / 332	0.92	13.19	105.82	119.01	

Table 4.2: Total running time of decomposing, unfolding and continuous folding in seconds.

for the head, 1 for the neck and 1 for the body. We then find the net and the folding path for each component separately. Average running time of 20 trails of each patch is shown in Table 4.1 from which we can see that nearly convex patch is more likely to be unfolded to a net that the start and goal can be linearly connected, thus the path planning time can be reduced significantly even in very high dimensional space. One thing worth noting is that since we can find different nets for each component, the shape and the topology of the nets, which determine whether start and goal are linearly connectable or not, have a huge impact on the difficulty of finding a feasible folding path. How to measure the goodness of a net in terms of its foldability is an interesting question and remains as future work.

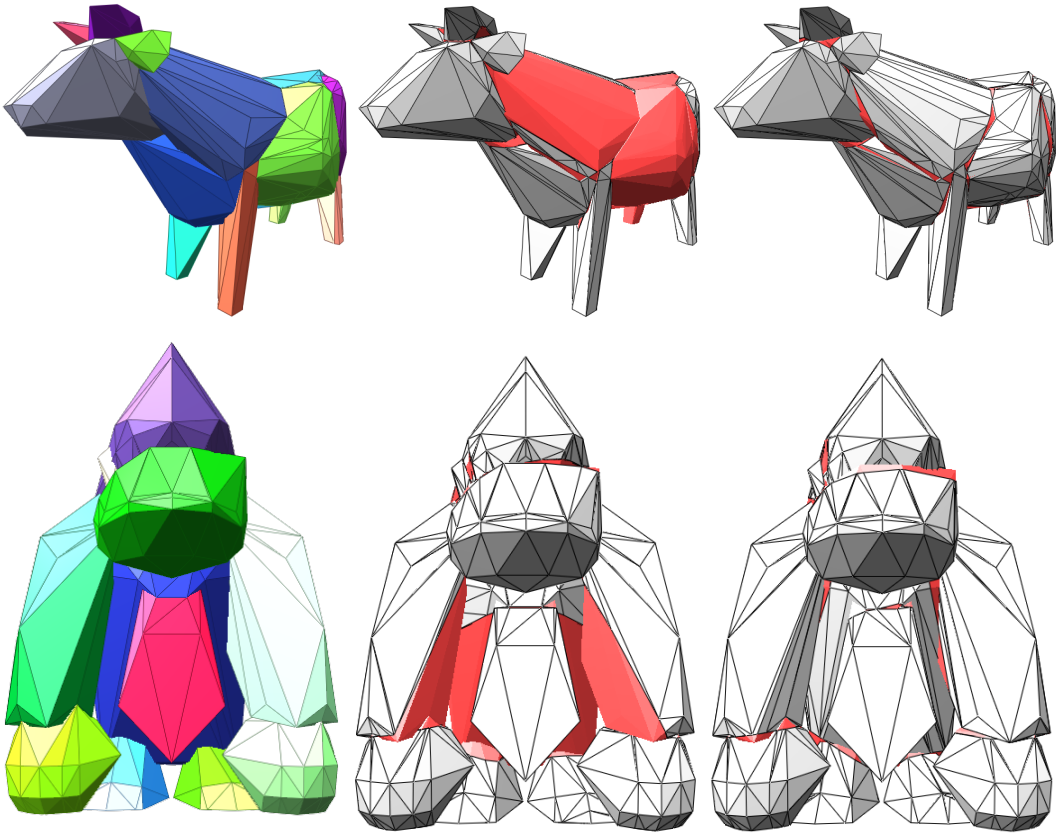
Chapter 5: Disjoint Convex Shell

5.1 Introduction

Approximating a 3D mesh by DC-shells enables more applications that approximating it using overlapping convex objects. Examples include faster and more robust collision response, better local penetration depth estimation, faster volumetric meshing and more realistic fracturing simulation. Without this constraint, the physical realization of the approximation is not even possible. We call this geometric structure, disjoint convex shell or simply **DC-shell**. The word *shell* is used to avoid confusion with convex hull.

There are several ways to produce DC-shell, including solid convex segmentation [43]. In this dissertation, we assume that the input shape is composed of multiple parts or is segmented either manually or algorithmically. Many meshes that are available in public sharing sites, such as *Thingiverse* or Google 3D warehouse, or those created for video games and animations come in this form. The convex hulls of the parts often overlap. Even under this somehow simplified setting, we will show theoretically that the *problem of converting overlapping convex objects to disjoint convex objects is difficult and computationally expensive*. Figure 5.1 illustrates failed attempts of a straightforward approach that trims the overlapping convex shapes using the least-squares fit of boundary intersections to find cut planes.

Throughout the work, we will use mesh unfolding to demonstrate the power of DC-shell. Mesh unfolding is an important computational problem in manufacturing a 3D shape from a 2D material [12, 13, 17]. Designing a foldable shape from a 3D mesh usually involves two main foldability analysis steps: instantaneous unfolding and continuous folding. Instantaneous unfolding instantaneously transforms a polyhedron to a 2D representation, which could be a net, a crease pattern or a developable surface. Suppose that a net is obtained,



(a) input (b) heuristics (c) optimized

Figure 5.1: **left**: Overlapping convex shapes depicting a cow (top) and Donkey Kong (bottom). **middle**: Cutting overlapping convex objects through their boundary intersection results in large volume loss shown in red. **right**: Disjoint convex shells created by our optimization method. The volume lost is significantly less.

the continuous folding problem aims to find a foldable path that transforms the net to its folded shape continuously without self-intersection. Both steps are significantly easier for convex shapes than non-convex shapes [3, 44].

In this disseration, we propose an optimization method for constructing DC-shells from a given composite shape. We also contribute a new remeshing method that enhances the quality of DC-shells while guarantees the convexity and disjointness of the DC-shells. Through its application in mesh unfolding, we show that DC-shells can be rigidly unfolded into high-quality nets with minimal cut length and coverage. We verify the foldability of these nets created from DC-shells algorithmically and through a user study with 102 school-age

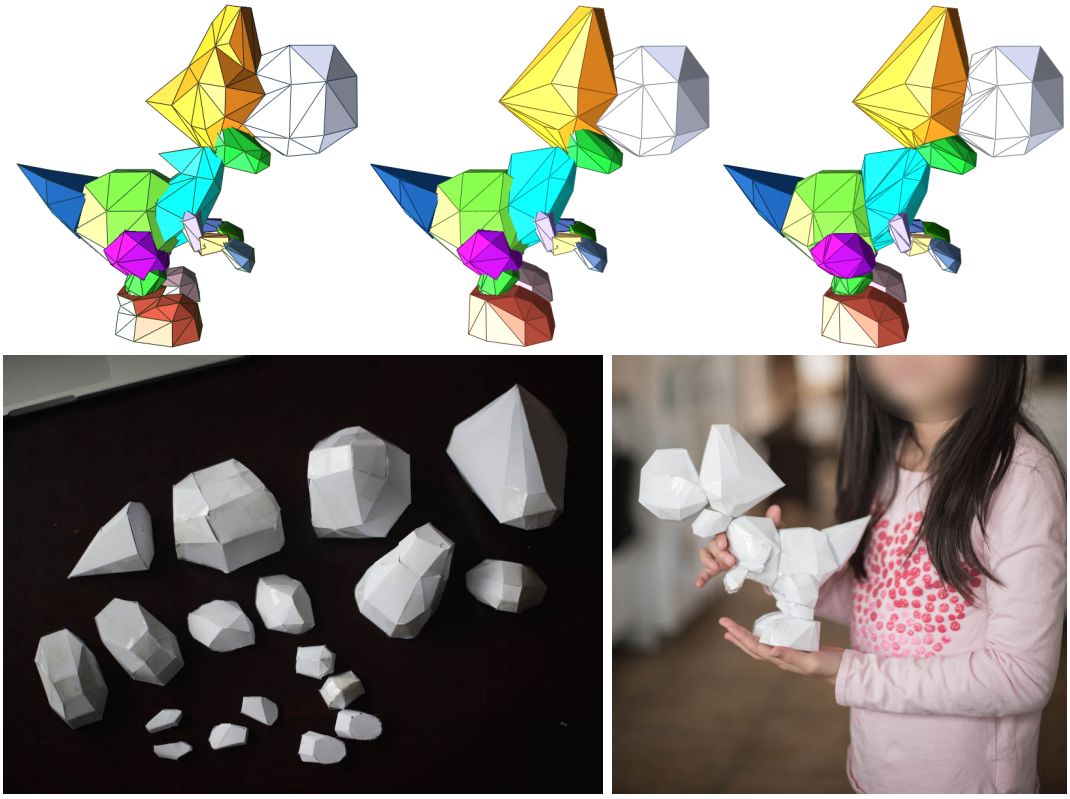


Figure 5.2: The top three figures, from left to right, show the composite shape of Yoshi model obtained from thingiverse.com, its convex hulls and its DC-shells. The bottom two photos show the folded disjoint convex shells and assembled Yoshi model.

children.

We show that children can work together and create rather complex 3D paper crafts around an hour in a hands-on story-telling class. Because these 3D paper crafts are made of blocks of convex objects, students can freely reconfigure their poses and even enlarge certain parts before assembly. Fig. 5.2 shows an example of a composite shape (Yoshi) obtained from thingiverse.com that contains overlapping parts and its DC-shells and fabricated paper model.

To the best of our knowledge, DC-shell is the first attempt to produce practical non-overlapping convex approximation. By practical, we mean the number of disjoint convex components is significantly smaller those created by existing methods. The most relevant work that we found is in computational geometry. There, researchers studied methods to

cover disjoint convex sets with non-overlapping convex polygons [45]. Our problem, on the other hand, is to find non-overlapping convex shapes approximating non-convex overlapping sets.

5.2 Related Works

5.2.1 Convex Decomposition and Approximation

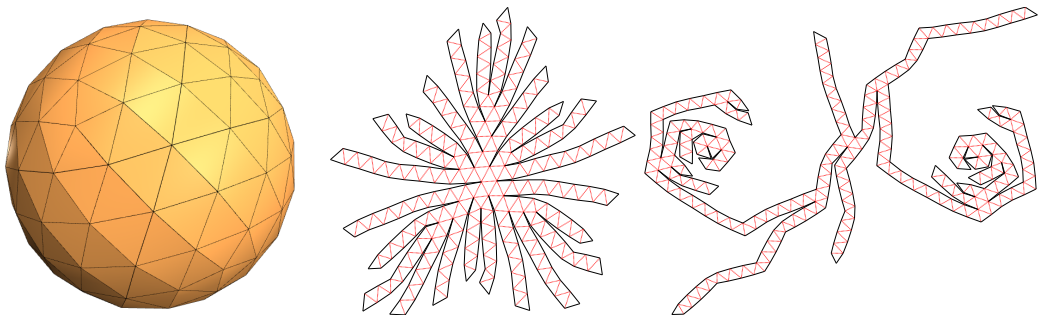
Convexity provides significant algorithmic benefits in many problems and motivates researchers to approximate shapes with convex objects. Covering a shape with non-overlapping convex shapes has been extensively studied. For example, exact convex decomposition [43, 46, 47] segments mesh into disjoint convex components. Another example is tetrahedral mesh generation. Both approaches can produce a large number of convex components. Exact convex decomposition is known to produce $O(r^2)$ convex components for a polyhedron with r reflex edges.

Approximations that allow overlapping convex shapes are more prevailing. For instance, approximating a shape with a set of primitive convex shapes, such as spheres, boxes, ellipsoids, and capsules, usually, forms a hierarchy of overlapping volumes. Another example is approximate convex decomposition (ACD) [32, 34, 48, 49, 50] that approximates the input mesh by a set of nearly convex shapes. Applications of ACD usually work solely with the convex hulls of the segmented parts and ignore the input mesh. These convex hulls usually overlap.

5.2.2 Polyhedra Unfolding

In the problem of unfolding the polyhedron to a 2D structure called the *net*, convexity admits simple edge unfolding of every convex polyhedron [44]. Finding a valid net of a given polyhedron is known to be nontrivial because a polyhedron with $|F|$ faces can have approximately $2^{\sqrt{|F|}}$ different unfoldings and most of them contain overlaps especially for non-convex polyhedra. However, if input shapes are convex polyhedra, heuristic methods

work well in practice. Most, if not all, nets of convex polyhedra with $||F||$ facets can be obtained in $O(|F|log|F|)$ time. However, it becomes much harder to generate a net for non-convex shapes. Mitani and Suzuki [17] decompose the mesh into a few patches and approximate each patch with a strip, a generalized cylinder or a developable surface. Schlickenrieder [3] proposed heuristics methods for unfolding a polyhedron to a net. Takahashi et al. [2], Straub and Prautzsch [16] extend [3] to unfold non-convex polyhedra. Recently, Xi et al. [19] proposed an approach that segments a model into a smell set of semantic and easily unfoldable parts by learning from failed unfoldings. All aforementioned works generate non-overlapping unfoldings, however, it is not guaranteed that there exists a continuous folding motion that transforms the net back to its original shape.



(a) Mesh (b) Net1 (c) Net2

Figure 5.3: A convex mesh its nets found by two heuristic methods: Steepest Edge for net1 and Flat Tree for net2 [3]. Both net1 and net2 were obtained within 0.01s. Net1's folding path is a straight line in the configuration space (only 1 edge was checked). For net2, in order to find a feasible path, 1056 edges need to be checked (approximately 1000 times slower) on average.

In order to make a physical copy of the foldable shape that can be continuously folded to its target shape from the net, we need to find a feasible folding path that can bring the net to its target shape without self-intersection. Again, if the input polyhedron is convex, the start state, i.e., the flat net and the goal state often can be linearly connected

in the configuration space. This property significantly reduces the path planning time to find a continuous folding motion. Fig. 5.3 shows a convex polyhedron and its nets, one of whose folding path is a straight line in the configuration space. However, when dealing with non-convex shapes, previous works [2, 8, 51] show that each step remains extremely challenging.

5.3 Building Disjoint Convex Shell (DC-shell)

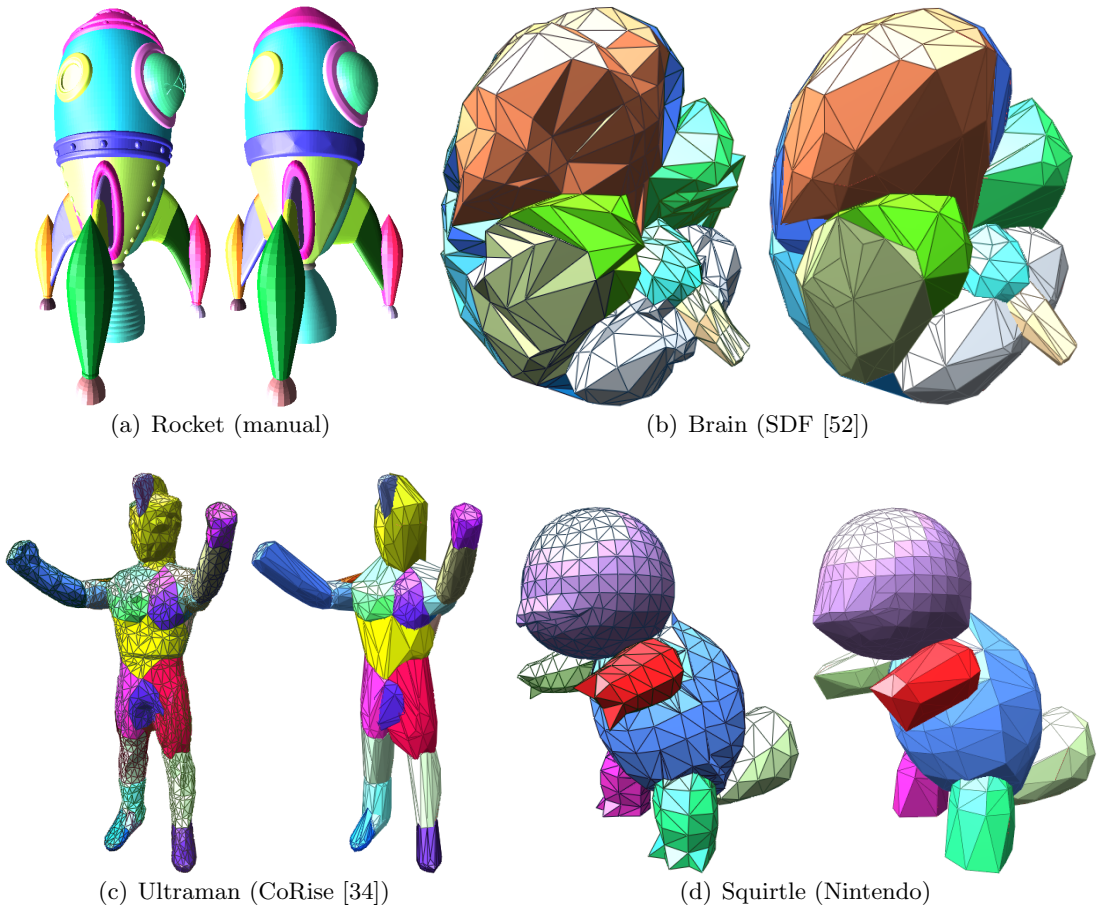
Given a composite shape P composed of components $\{P_i\}$, which may either overlap or overlap at boundaries with disjoint interiors. Fig. 5.4 shows examples of composite shapes obtained from various sources. Given a pair of adjacent components P_i and P_j, the intersection of their convex hulls $C_i \cap C_j$ is usually not empty.

A first important observation is that the separator of two overlapping convex shapes C_i and C_j mush be linear. If the separator is curved, one of the separated shapes must be non-convex. Given that the separator must be a plane, it is desirable to constrain the cutting plane $\mathcal{H}$ so that the disjoint shapes are separated without a gap in between (i.e. with overlapping boundary on $\mathcal{H}$). Fig. 5.5 illustrates this idea. To ease our future discussion, we will use the term 'trim' to describe the operation of removing a small volume of an object by interesting the object and the closed half space $\mathcal{H}^+$ or $\mathcal{H}^-$ bounded by $\mathcal{H}$. We will also use C'_* to denote the trimmed shape, i.e., $C'_i = C_i \cap \mathcal{H}^+$ and $C'_j = C_j \cap \mathcal{H}^-$.

In the rest of this section, we will start with a heuristic that finds the cutting plane via least squares fitting and discuss its limitations. To address these limitations, we formulate an optimization problem and show that solving this optimization problem efficiently is challenging even in the discrete domain.

5.3.1 A Heuristic using Least Squares Fitting

Naturally, our first attempt computes the intersection of convex hull boundaries ∂C_i and ∂C_j and the cut plane $\mathcal{H}$ least-squares fits the intersection. Because the cut plane $\mathcal{H}$ fits the



(a) Rocket (manual)

(b) Brain (SDF [52])

(c) Ultraman (CoRise [34])

(d) Squirtle (Nintendo)

Figure 5.4: Examples of composite shape used in the experiments including models created by manual segmentation (a), part-aware decomposition (b)&(c), and models composed of multiple overlapping parts (d). In all these examples, the convex hulls of their parts overlap.

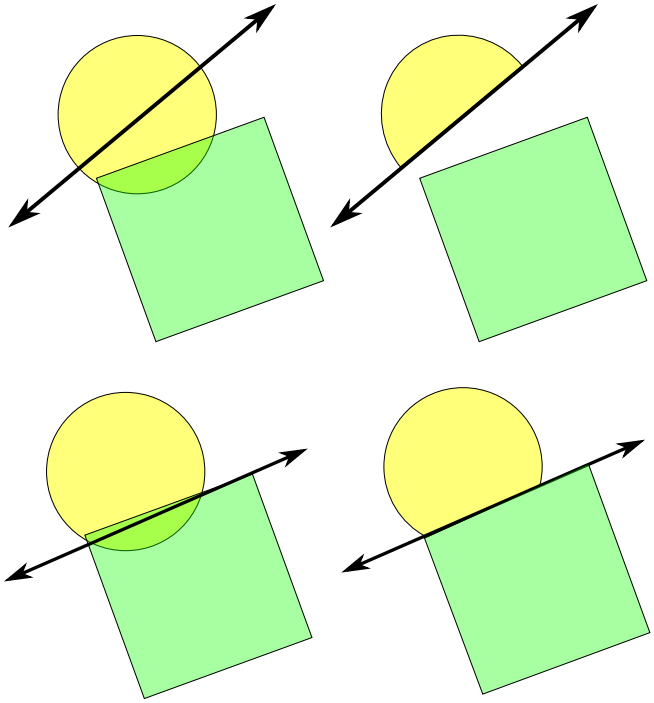


Figure 5.5: **Top**: The cut results in a gap after trimming. **Bottom**: Overlapping boundary with disjoint interior is more desirable.

surface intersection, it is guaranteed that the trimmed shapes C_i' and C_j' is interior disjoint but exterior connected without any gap. In many cases, this simple method creates decent results as shown in Fig. 5.6.

Although this least-squares fitting approach is extremely fast and easy to implement, its quality is unbounded. In particular, the volume trimmed by the cut plane is not considered as shown in the examples in Fig. 5.1. In the cases that ∂C_i and ∂C_j intersect at coplanar or nearly coplanar facets, have multiple connected-component intersection due to extrusion, or have highly non-planar intersection, least-squares fitting is too local and greedy and is clearly not a good approach. Fig. 5.7 shows two examples of such complications.

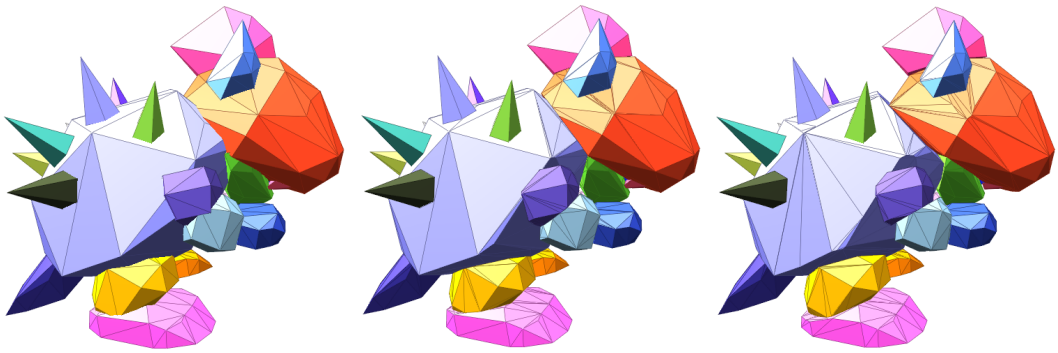


Figure 5.6: **left**: overlapping convex hulls of a Bowser model. **middle**: DC-shells created using least-squares fitting heuristic. **right**: Results of proposed method.

5.3.2 Disjoint Convex Shells

To provide higher quality disjoint convex shells, it is desirable to minimize the volume of the underlying volume removed by plane $\mathcal{H}$ in $\mathbb{R}^3$, therefore $\mathcal{H}$ should be derived from the following optimization function:

$$\arg\min_{\mathcal{H}} \left(\text{vol}(P_i \cap \mathcal{H}^+) + \text{vol}(P_j \cap \mathcal{H}^-)\right) \ , \tag{5.1}$$

where $\mathcal{H}^+$ and $\mathcal{H}^-$ are the half-spaces on the positive and negative sides of $\mathcal{H}$. Because P_i and P_j not only are non-convex but also often contain surface holes and are non-manifold, the optimization problem in Eq. 5.1 becomes ill-defined for many composite models. Instead, we will focus on the following problem:

$$\arg\min_{\mathcal{H}} \left(\text{vol}(C_i \cap \mathcal{H}^+) + \text{vol}(C_j \cap \mathcal{H}^-)\right) \ . \tag{5.2}$$

In addition, to ensure that no gap is introduced by $\mathcal{H}$, $\mathcal{H}$ must pass through a point in $C_i \cap C_j$. Let $o_{\mathcal{H}}$ be a 3D point on $\mathcal{H}$ and $n_{\mathcal{H}}$ be the normal direction of $\mathcal{H}^+$. The constrains ensure that $o_{\mathcal{H}} \in C_i \cap C_j$ and the objective function determines the exact position $o_{\mathcal{H}}$ and orientation $n_{\mathcal{H}}$ so that the trimmed volume is minimized.

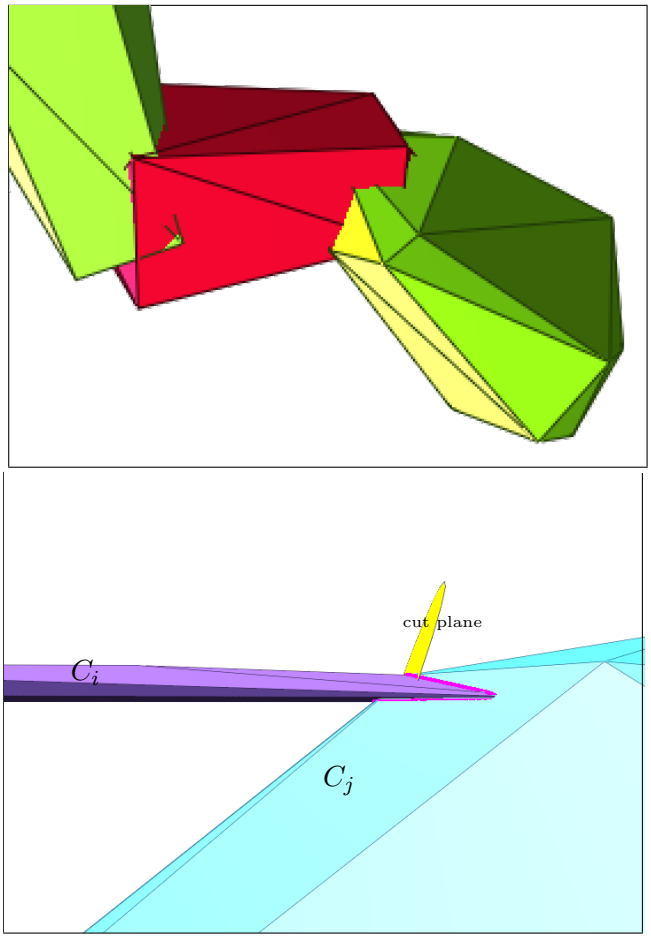


Figure 5.7: Complex near-coplanar overlap and extrusion between two convex objects

The key to solving the aforementioned optimization problem is a formulation expressing the volume of the intersection between a known convex object and unknown 3D plane. In the literature, it is known that the function of the volume of a convex object intersected by a moving half-space is highly non-linear even in $\mathbb{R}^3$. In fact, computing the exact volume of a convex shape intersected by a translating plane in arbitrary dimension is known to be #P-space hard [53, 54, 55]. Several approximation methods were introduced [54, 56]. Almost all of these methods adopted the idea of uniform sampling of points inside the convex object [57]. The idea is that, if points are sampled uniformly inside the convex shape, the number of points approximates the volume. Naturally, we will start off our discussion by assuming

that the volumes are approximated by samples in C_i and C_j.

Approximating the Volume with Samples

In this section, we will present two methods that use uniform samples to determine the separating plane. It is important to note that both methods can also handle points that are created from the segmented components P_i even though our discussion assumes that points are sampled from C_i.

Let S_i and S_j be two point sets in $\mathbb{R}^3$ whose convex hulls overlap. We assume that S_i and S_j are uniformly distributed in a subspace of $\mathbb{R}^3$, such as the space enclosed by their convex hulls. The objective is to find a plane $\mathcal{H}$ that minimizes the number of points from the same set to be separated by $\mathcal{H}$.

$$\arg\min_{\mathcal{H}} \left(||S_i \cap \mathcal{H}^+|| + ||S_j \cap \mathcal{H}^-|| \right) \ , \tag{5.3}$$

where $||X||$ is the cardinality of a finite set X.

We first show that finding all optimal solutions of $\mathcal{H}$ takes $O(||S_i||^3||S_j||^3)$ by reduction from the classic ham sandwich problem.

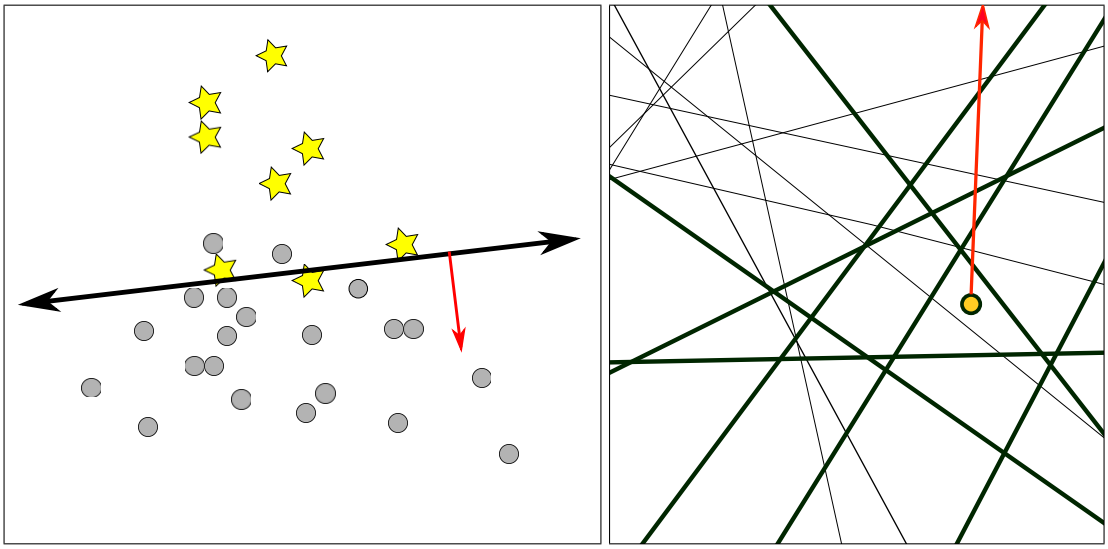
Arrangement of Dual Points

Duality has been used in problems such as linear programming. In this method, we will use an idea similar to that solves the *ham sandwich problem* [58, 59, 60]. That is, we will find the plane $\mathcal{H}$ in the dual space in which a 3D point $P^* = (a, b, c)$ represents a 3D plane P in primal space with normal direction $(a, b, \sqrt{1 - a^2 - b^2})$ and offset c from the origin. Similarly, a point $Q = (a, b, c)$ in the primal space becomes a 3D plane Q^* in the dual space. The problem of finding $\mathcal{H}$ is then equivalent to finding a point $\mathcal{H}^*$ such that the sum of the number of planes S_i^* below $\mathcal{H}^*$ and the number of planes S_j^* above $\mathcal{H}^*$ is minimized. Fig. 5.8 illustrates this idea in 2D. Next we show that finding all optimal cuts takes $O(||S_i||^3||S_j||^3)$ time.

Theorem 5.1. Given two finite point sets S_i and S_j whose convex hulls overlap, finding

all optimal cuts of Eq. 5.3 takes $O(||S_i||^3||S_j||^3)$ time.

Proof. To solve this dual version of our optimization problem, one will have to compute two arrangements $\mathcal{A}(S_i^*)$ and $\mathcal{A}(S_j^*)$ of the planes S_i^* and S_j^*, respectively. In each arrangement, every facet will be annotated with the number of planes blow the facet, called *level* [58]. It is easy to show that every point on the same facet has the same level. Computing such arrangement and annotating its facets will take $O(n^3)$ time where n is the number of points in S_i or S_j. Finally, we overlap $\mathcal{A}(S_i^*)$ and $\mathcal{A}(S_j^*)$, and the solution reported as a pair of intersecting facets $(f_i \in \mathcal{A}(S_i^*), f_j \in \mathcal{A}(S_j^*))$ such that level of f_i minus the level of f_j is minimized. $\mathcal{H}^*$ must be a point on the intersection of f_i and f_j. This leads to a computation of the intersections of $\mathcal{A}(S_i^*)$ and $\mathcal{A}(S_j^*)$, which takes $O(n^3m^3)$ time, where n and m are the number of points in S_i and S_j, respectively. □



(a) primal (b) dual

Figure 5.8: (a) The problem of finding a cutting line that minimizes the sum of the number of stars below the line and the number of circle above the line. (b) The problem of (a) is equivalent to finding a point so that the sum of the number of thick lines (dual of the circles) vertically below and the number of thin lines (dual of the stars) vertically above the point is minimized.

Unfortunately, this approach, that finds all possible solutions of $\mathcal{H}$, is impractical since we only need one of such $\mathcal{H}$. In the next section, we investigate a more efficient approach using Support Vector Machine.

Using Support Vector Machine

Support Vector Machine (SVM) [61] is a widely used classifier due to its solid mathematic background, its fast computation speed and high classification accuracy. SVM finds a linear hyperplane to separate positive and negative data in feature space. The hyperplane is parameterized by $\mathbf{c}$ and d, where $\mathbf{c}$ is the normal vector and d is the offset of the hyperplane. The distance from the closest point to the hyperplane is called *margin*. SVM minimizes $||\mathbf{c}||^2 + \mu \sum_i \zeta_i$ where μ is user parameter and ζ_i is classification error, subject to the constraint:

$$y_i(\mathbf{c}^T \mathbf{x}_i + d) >= 1 - \zeta_i \ ,$$

where $y_i \in \{-1, 1\}$ indicates the label of $\mathbf{x}_i$. This is known to be a quadratic programming problem and can be solved efficiently in most cases.

Given point sets S_i and S_j, finding the cut plane using SVM to minimize the Eq. 5.3 is straightforward as we do not consider nonlinear kernel functions and the dimensionality of our problem is low comparing to the dimensionality of typical classification problems solved by SVM. However, it is worth noting that, in our problem of separating two convex shapes, the penalty parameter μ plays a critical term as in majority of cases, the point sets S_i and S_j are not linearly separable and therefore the main task is to minimize the loss function $\sum_i \zeta_i$ that is proportional to the distance of misclassified points to the margin.

This becomes problematic because the penalty parameter μ might need to be different for different models and even between different parts within the same model. For models whose convex hulls are linearly separable, small μ is sufficient. For models with significant overlap between convex models, in particular when two overlapping convex shapes have significant difference in size (i.e., one convex shape is much larger than the other one), μ needs to be very large to ensure correct separation. Fig. 5.9 shows examples of different

μ creating different results. Sometimes, the gaps cannot be eliminated even when μ is as large as ten thousand. In addition, as μ increases, the optimal solution becomes harder to find and SVM takes much longer time to coverage and often fails to converge. Solutions to handle such sampling discrepancy for classification problems usually involve assigning weights to samples or increasing samples. Both approaches are not viable to us because the size of samples represents volume.

It might also be tempting to separate only the points in $C_i \cap C_j$, instead of all points sampled, to avoid the aforementioned pitfalls. However, again, this local and greed approach will create undesirable results in which the cut-off volume becomes unbounded.

Exact Volume Computation

The issue of the penalty parameter μ can be addressed if we can be less dependent on the sampling discrepancy. The volume of the intersection between a convex shape $C = \{x \in \mathbb{R}^n : b_i - \mathbf{a_i}^T x \geq 0\}$ and a translating plane $\mathcal{H}(x) = \mathbf{c}^T x + d$ is known to be the sum of a linear function between the vertices of C and $\mathcal{H}(x)$ [55]:

$$V_C = \sum_{v \in C} \frac{\max(0, H(v)^n)}{n! \delta_v \gamma_1 \cdots \gamma_n} \ , \tag{5.4}$$

where γ_i satisfies $\mathbf{c} = \sum_i \mathbf{a_i} \gamma_i$ and δ_v is $|det([\mathbf{a_1} \cdots \mathbf{a_n}])|$, and $\mathbf{a_i}$ is the normal of a hyperspace of C containing v.

Computing Eq. 5.4 in general space is #P-space hard mainly due to the curse of dimensionality and the vertices v are unknown. Fortunately, our problem is in 3 space and all the vertices of C are known. In 3D, the volume can be simplified to

$$V_C = \sum_{v \in C} \frac{\max(0, \mathcal{H}(v)^3)}{6 \delta_v \gamma_1 \gamma_2 \gamma_3} = \sum_{v \in (C \cap \mathcal{H}^+)} \frac{(d + \mathbf{c}^T v)^3}{\Delta_v} \ . \tag{5.5}$$

Now, if we let $\mathcal{H}$ translate, then V_C is a function of d. Given two overlapping convex

shapes C_i and C_j, our goal is to find the offset d such that $D = V_{C_j} - V_{C_i}$ is minimized[1]. To determine d that minimizes D between a pair of consecutive vertices of C_i and C_j, we simply find the roots of the following quadratic equation $\frac{\partial D}{\partial d} = 0$, i.e.,

$$\sum_{v \in (C_j \cap \mathcal{H}^+)} \frac{3(d + \mathbf{c}^T v)^2}{\Delta_v} - \sum_{u \in (C_i \cap \mathcal{H}^+)} \frac{3(d + \mathbf{c}^T u)^2}{\Delta_u} = 0 \ . \tag{5.6}$$

Fig. 5.10 compares the DC-shells created using SVM with and without minimizing D. The volume loss is decreased by half using exact volume computation.

Note that, it may be tempting to discard SVM entirely and find the cut plane $\mathcal{H}$ by minimizing Eq. 5.6 with the full degrees of freedom. This means, $\mathbf{c}$ is also unknown, and we will need to solve three first-order partial derivatives $\frac{\partial D}{\partial \mathbf{c}_x} = 0$, $\frac{\partial D}{\partial \mathbf{c}_y} = 0$ and $\frac{\partial D}{\partial d} = 0$. However, solving $\frac{\partial D}{\partial \mathbf{c}} = 0$ is very challenging as $\mathbf{c}$ appears in both nominator and denominator of D, thus requiring us to find roots of three degree-4 polynomials with multiple degree-6 polynomial constraints. This will require further investigation in our future research.

Handling Interference Between Cuts

Our discussion so far has centered around the case of pairwise overlapping. When more than two convex objects overlap sharing a common region, these cut planes can have mutual interference. To describe this phenomenon more clearly, we first observe that: A cut plane $\mathcal{H}_{ij}$ between C_i and C_j can only affect the other overlapping pairs C_i and C_k if $\mathcal{H}_{ij}^-$ encloses $C_i \cap C_k$. When this happens, we say that $\mathcal{H}_{ij}$ and $\mathcal{H}_{ik}$ interfere mutually.

Notice that, this observation implies the scenarios of three or more overlapping objects sharing a common region. Moreover, this observation also implies that interference can only be realized through a shared object, i.e., C_i in our example.

It is important to note that under these interferences, our objective remains the same:

[1]We actually wanted to minimize $V_{C_j} + \text{vol}(C_i) - V_{C_i}$, but $\text{vol}(C_i)$ is a constant so it is equivalent to minimizing $V_{C_j} - V_{C_i}$.

minimizing volume loss. Therefore, the cut plane $\mathcal{H}_{ij}$ interfering $\mathcal{H}_{ik}$ should still trim C_i and C_j as little as possible and, at the same time, it should also minimize the amount of interference defined as the volume of $\mathcal{H}_{ij}^{-} \cap (C_i \cap C_k)$. This additional requirement can be added to Eq. 5.6 to find a cut plane $\mathcal{H}$ between C_i and C_j that minimizes it inference to all affected C_k:

$$\frac{\partial D}{\partial d} + \lambda \frac{\partial (\sum_k \text{vol}(C_k \cap C_{i \vee j}, \mathcal{H}^{-}))}{\partial d} = 0 \ , \tag{5.7}$$

where λ is a user defined weight that penalizes interference between $\mathcal{H}$ and the pair C_k and C_i or C_k and C_j. In all experiments reported in this disseration, we chose a small $\lambda = 0.01$. Eq. 5.7 can be solved using the same strategy discussed in the previous section as $C_k \cap C_{i \vee j}$ is also convex. Fig. 5.12 shows before and after inference is penalized.

5.4 Convex Shell Simplification and Regularization

Convex shapes considered in this dissertation are created by taking the convex hulls of composite shape. Consequently, their facets tend to be skinny and often densely packed near certain spots. Before these convex objects are trimmed, it is often desirable to simplify and remesh them to decrease the complexity and increase their regularity. The trimming process discussed earlier may often produce many small and skinny sliver triangles. It is critical that these slivers can be removed. Remeshing also makes unfolding and folding easier, thus ease the fabrication process.

There have been extensive study of convex and non-convex polyhedra simplification and remeshing, e.g., see reviews in [62]. In our case, we require that the simplified and remeshed shape maintains its convexity and interior disjointness. We further require that the simplified and remeshed convex shape must enclose the original convex shapes to avoid introducing new gaps. Finally, under these critical constrains, we would like to minimize the gained volume after remeshing.

Simplification. The progressive hull introduced by Sander et al. [63] can be easily

adopted for our requirements. The approach is based on a sequence of edge contractions, in which an edge is contracted to a vertex. The vertex is placed according to a constrained optimization problem. The constraints are that the vertex must be placed inside the cut planes and outside the plane of every face incident to the edge. Equivalently, every tetrahedron formed by connecting the new vertex to a face incident to the edge must add volume to the shape. The progressive hull objective function minimizes the total added volume. The next edge to contract is the one whose contraction adds the least volume. Edges with no solution satisfying the constraints are skipped. The constraints and objective function are linear, since the signed volume of a tetrahedron with three fixed vertices is linear:

$$\frac{A}{3} n \cdot (v - v_0) \ ,$$

where v is the free vertex, v_0 is any one of the fixed vertices, and n and A are the outward unit normal and area of the triangle formed by the three fixed vertices. As a result, the constrained optimization for each edge contraction is a small linear programming problem that can be solved efficiently.

Regularization. Although many remeshing methods exist including isotropic remeshing [64] and as-equilateral-as-possible remeshing [65]. To the best of our knowledge, no remeshing method can ensure convexity. In fact, our study also finds that, while some vertices are outside the original convex shape, most vertices are inside. This introduces gaps between parts after remeshing. An example of such problems is shown in Fig. 5.13(c). To regain convexity, Fig. 5.13(d) shows the convex hulls of Fig. 5.13(c) but regularity is destroyed.

We propose a new method to address this problem using Laplacian smoothing [66] that moves every vertex to the center of its neighbors. We formulate the remeshing problem as a sequence of quadratic programming problems. Each quadratic programming problem minimizes the distance between a vertex v and the center of its neighbors with linear constrains, similar to the aforementioned constraints for progress hulls construction, ensuring that the

mesh remains convex and disjoint after moving v. A result obtained from the proposed method can be found in Fig. 5.13(b). An extensive study of this new method can be found in Section 5.6.3.

5.5 Unfolding and Folding DC-shells

While DC-shell can be used in many other domains (such as fracturing shown in the supplementary video), our work on creating DC-shell is motivated by mesh folding and unfolding. Although efficient algorithms exist for unfolding convex shapes [3], they did not consider the quality of nets (e.g. ease of fabrication and foldability). In this dissertation, we extend a genetic-based algorithm [19] to optimize the nets of DC-shells using the fitness function $f = \sum_i w_i x_i$, where x_i is the feature value and w_i is its corresponding weight.

Convex hull area and total cut length are two features considered here. Intuitively, a smaller convex hull area means the net has a tighter representation which reduces the fabrication time; it also indicates the net has fewer long branches that are harder to fold. Since all cut edges need to be glue together when folded, shorter total cut length also makes folding more easier and makes the foldable object mechanically more stable. In Fig. 5.14, we show an example of net optimization, from which we can see that, initially, the net (Fig. 5.14(b)) contains a lot of skinny triangles like spines. After optimization, we get more desirable nets (Fig. 5.14(c) and Fig. 5.14(d)) whose convex hull area or total cut length reduced by more than 40%.

5.6 Experimental Results

We implemented the proposed DC-shell methods in C++, which will be released to the public domain. All data reported here were collected on a 2015 MacBook Pro with a 2.5 GHz Intel Core i7 CPU and 16 GB Memory.

Model	Triangles	**LSF**	**SVM**	**Exact**
Bowser	999	4.153	18.708	16.391
Brain	1209	2.553	8.811	9.018
Bulbasaur	4971	5.440	10.673	11.705
Bull	105988	17.584	28.386	27.127
Chicken	5000	5.070	12.087	12.479
Cow	5804	7.575	14.257	14.888
Crocodile	5250	6.065	9.922	9.770
Dancing Children	3000	8.898	10.665	11.451
Donkey Kong	900	2.428	18.224	19.557
Frog	53006	8.841	12.091	13.734
Haechi	5001	4.878	12.034	11.940
Hover bike	9331	4.290	45.018	52.604
Mother	996	4.006	8.002	8.306
Pikachu	4368	5.504	7.866	8.017
Rabbit	61026	15.658	23.398	23.221
Rocket	254541	55.131	61.148	59.383
Squirtle	1718	3.401	6.051	6.307
Ultraman	4999	8.140	10.300	11.012
Vulpix	1934	4.932	6.409	7.125
Yoshi	677	3.609	11.730	12.583
union	1525	0.039	0.039	0.038

Table 5.1: Running Time (seconds)

5.6.1 Running Time

Table 5.1 reports the running times of our implementation of all three methods discussed in Section 5.3, namely LSF (least-squares fit) heuristic method, SVM and exact volume optimization methods. In our experiments, we simplify and regularize the meshes 10 iterations with maximum volume increase ratio less than 5% of the original volume using the method discussed in Section 5.4. According to Table 5.1, LSF is fastest (including simplification and regularization time) but the proposed optimization methods using either SVM or exact volume computation are only 2 to 10 seconds slower, except for the "Hover bike" model. SVM and exact methods are much slower in this example because Hover bike contains 49 components and many of them are small and deeply embedded in some larger convex parts.

5.6.2 Quality Comparison

The quality of DC-shell is measured by volume loss from trimming the input convex shapes. That is to say, a DC-shell has higher quality if it removes less material from input to provide disjointness. More specifically, volume loss is defined as

$$\frac{\text{vol}_{orig} - \text{vol}_{shell}}{\text{vol}_{orig}} , \tag{5.8}$$

where vol_{orig} is the volume of the union of the original convex shapes and vol_{shell} is the sum of the volume of all disjoint convex shells.

Case study. We start by measuring the volume loss of the proposed methods with models obtained from various sources. Table 5.2 reports the volume loss in percentage of all DC-shells generated from least-squares fit, SVM, and the exact methods. As the original convex objects are not disjoint, to obtain the original volume, we need to perform boolean union. CGAL is used to perform this operation. In the case that CGAL failed to create a 2-manifold mesh, a volumetric method is used to provide a tight upper bound of the union. The least-squares fit (LSF) method performs quite well on some models such as Squirtle but extremely poor on the others such as Bull model with over 24% volume loss. The SVM method has smaller volume loss than LSF in most cases but performed worse than LSF on Bowser, Spaceship, and Squirtle models and still has over 10% volume loss on the Bull and Hover bike models. Finally, the exact method consistently provides the highest quality DC-shell in all models experimented. Even fro the cases that LSF and SVM methods perform well, the exact method can always reduce volume loss even further.

Controlled study. In this experiment, two intersecting convex shapes have various degree of overlapping. Fig. 5.15 shows the DC-shells generated. LSF method consistently generates low quality results with large volume loss (11.43% on average) due to none-planar local intersection. The optimization methods using SVM and exact volume computation, on the overhand have only 2.1% and 0.9% volume loss, respectively. In general, SVM and

	Orignal volume	**LSF**	**SVM**	**Exact**
Bowser	1112400	0.37%	**0.59%**	0.29%
Brain	508.462	**2.69%**	1.32%	1.17%
Bulbasaur	883179	0.15%	**0.18%**	0.14%
Bull	1.91992	**24.67%**	10.76%	2.12%
Chicken	142277	0.10%	**0.18%**	0.10%
Cow	110099	**11.93%**	1.32%	1.03%
Crocodile	27460.2	**33.09%**	4.46%	4.40%
Dancing Children	4739.83	**3.00%**	1.62%	0.86%
Donkey Kong	332858	**22.48%**	1.90%	1.73%
Frog	93009.3	**1.83%**	0.24%	0.24%
Haechi	291371	**0.73%**	0.64%	0.62%
Hover bike	159621	6.70%	**10.62%**	5.79%
Mother	295009	**0.74%**	0.49%	0.31%
Pikachu	80684.7	**0.181%**	0.102%	0.096%
Rabbit	366139	**1.36%**	0.84%	0.67%
Rocket	3684920	0.84%	**4.2%**	0.40%
Squirtle	98.6569	0.25%	**0.45%**	0.22%
Ultraman	297887	**11.24%**	1.13%	1.07%
Vulpix	247.295	0.64%	**0.65%**	0.42%
Yoshi	576266	0.56%	**0.69%**	0.47%

Table 5.2: Quality of DC-shells produced using least squares fit (LSF), SVM and exact methods is measured by volume loss defined in Eq. 5.8. The penalty parameter C is 100 for all examples. The method with the largest volume loss for each model is shown in bold.

exact methods produce similar results, except the last example where two bars overlapping vertically. The exact method produces more natural DC-shells with smaller volume loss.

5.6.3 Results from Convex Remeshing

The quality of regularization is measured by the weighted average fatness of the mesh facets. Fatness of a given triangle f is defined as $\cos\angle a + \cos\angle b + \cos\angle c - 1$, where $\angle a$, $\angle b$, $\angle c$ are internal angles of f. The optimal fatness is 0.5. Then weighted average fatness of a given polyhedron P is $\frac{\sum_{f\in P} A_f \text{fatness}(f)}{\sum_{f\in P} A_f}$, where A_f is the area of f.

Fig. 5.16 shows weighted average fatness from 1 to 100 iterations of remeshing at four different levels of maximum volume increase percentages: 1%, 10%, 100%, and 1000%.

Larger maximum volume increase percentages allows more room for optimization, thus should provide better regularization. From the plots, we can see that, with 1% maximum volume increase, the facet fatness increases gradually. At 10% to 1000% maximum volume increase, the fatness increases drastically and barely change after it reaches a certain fatness. The weighted average fatness increase by approximately 25% at least after regularization. The plots in Fig. 5.16 also shows the fatness of the convex hulls of remeshed convex objects using isotropic remeshing implemented in CGAL, i.e., Fig. 5.13(c) and (d). The fatness of convex shapes obtained through this approach are consistently lower than ours.

5.7 Fabricating Physical Models

This section details the application of DC-shell in making paper craft. Composite shapes used in the experiment are shown in Fig. 5.17. Although not demonstrated here, we believe DC-shell can also facilitate other manufacturing process, including traditional subtractive manufacturing (e.g., wood carving) and modern additive manufacturing methods (e.g., 3d printing).

Generating Foldable Nets. We first unfold the original composite shape and its DC-shells into nets. For fair comparison, we simplify the original model such that the number of triangles equals to total number of triangles of all DC-shells. Once we obtained the optimized nets, motion planning technique [11] is adopted to find a folding motion that continuously transform the net from the flat state to the folded state. Continuous motion is critical which ensures the model is physically realizable, besides, it can also act as a folding guide for human folders. The computation time of unfolding and folding all components is reported in Table 5.3, from which we can see that DC-shells significantly reduce the unfolding and path planning time by several orders of magnitude. It is important to note that some parts of the composite shapes cannot be unfolded.

Fabrication and Comparison. We compare the folding times required to fold the original shape and DC-shells using three models from Fig. 5.17 whose original mesh can be unfolded successfully. Recall that the the original shape and DC-shell have the same

	Bulbasaur	Chicken	Bull	Crocodile	Haechi	Rabbit	Frog	Squirtle
Number of triangles	1138	1028	1038	1570	608	936	610	542
Number of parts	12	12	12	17	6	10	6	7
Patch Unfolding Time (s)	112.384	469.288*	462.868*	684.161*	174.831*	95.717*	36.122	15.179
Patch Folding Time (s)	72.468	541.273*	201.821*	89.394*	4.442*	63.931*	294.184	9.584
DC-shell Unfolding Time (s)	0.065	0.068	0.074	0.124	0.032	0.059	0.045	0.034
DC-shell Folding Time (s)	1.255	1.382	1.478	2.031	0.843	0.985	0.732	0.426

Table 5.3: Folding and unfolding time in seconds of segmented parts and their DC-shells. Running times marked with $*$ indicate certain parts of the original models have no valid nets.

number of facets. Details on the tools and fabrication steps can be found in Supplementary materials. Fabricated results are shown in Table 5.4 and Fig. 5.18. The subjects both commented that nets of DC-shell are easier to fold comparing to that of original mesh patches, and DC-shells are easier to assemble since they share cutting planes while nearly convex patches only share cutting edges.

	Bulbasaur	Frog	Squirtle
Original mesh	118 mins	104 mins	90 mins
DC-shells	80 mins	59 mins	60 mins

Table 5.4: Fabrication time by two adult subjects

User study. We conducted a user study with 102 children at an elementary school whose ages are 9 and 12 years old during class hours. They worked together in groups of three or four. Depending on their lessons, students fold a subset of models from Fig. 5.17. Upon completion, students then decorated their finished craft for a puppet show.

To ensure that the students can complete their folding on time, we enlarged some small parts (e.g., horn, ear, beak). We also added square marks and short strokes on to the nets that represent which vertex is a starting point to fold nets and which edges are assembled, respectively. These two types of marks help students fold without detailed instructions. We taught students the meaning of these marks and ask them to fold each net into a convex shape.

The folded models are shown in Fig. 5.19. The study didn't show significant differences in fabricating time between 9-year-old and 12-year-old students in Table 5.5. The younger students did the paper craft as well as the older students did, although Bulbasaur, chicken and bull models have more than 1000 triangles. This indicates that DC-shell eases the fabrication process. In addition, the crocodile model has 2.9 times and 1.7 times more triangles that those of Squirtle and Rabbit, nonetheless, the difference in fabricating time between the crocodile model and the others is only 1.1%.

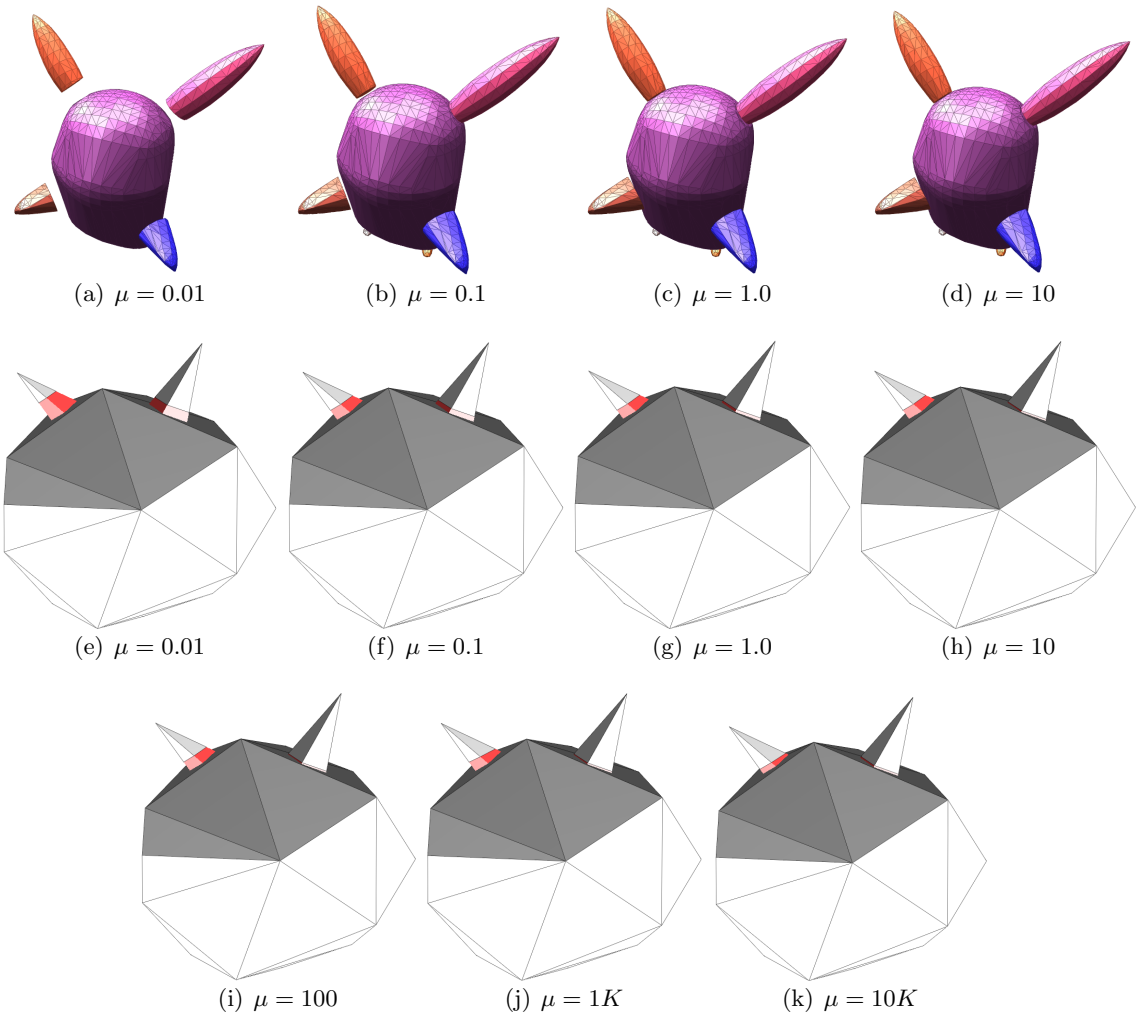
Model	Bulbasaur	Chicken	Bull	Crocodile	Haechi	Rabbit	Frog	Squirtle
Number of triangles	1138	1028	1038	1570	608	936	610	542
Fabricating time (number of students)	47 (3) 65 (3) 65 (3) 75 (3) 40 (3*)	35 (3) 50 (3) 55 (3) 45 (3) 45 (3*)	50 (3) 43 (3) 40 (3) 48 (3) 60 (3*)	65 (4) 60 (3) 55 (3)	63 (3) 41 (3) 50 (3) 33 (3*)	48 (3) 62 (3) 57 (3) 74 (3) 35 (3*)	50 (3) 20 (3)	49 (3) 60 (3) 60 (3) 50 (3) 55 (3*)
Average time	58.4	46	48.2	60	46.75	55.2	35	54.8

Table 5.5: Fabricating time in minutes. Data collected from groups of three or four 9 and 12-year-old students at an elementary school. * test conducted by 12-year-old students.

5.8 Conclusion

In this chapter, we presented an optimization method that creates pairwise disjoint convex sets from a composite shape. We showed that this seemingly simple problem is in fact computational expensive and simple solutions do not work. Our methods combined SVM and exact volume computation to find cut planes that minimize volume loss from trimming the overlap. The quality of the proposed methods was shown experimentally better than heuristic methods using composite shapes from various sources. Mesh unfolding was used to further demonstrate the benefits provided by DC-shells via virtual (algorithm) folders and human folders. Our user studies showed that DC-shells made paper craft creation and design more accessible to school-age children and provides chances to enrich their education experiences. One major limitation of our approach is its dependency on composite shapes.

Future work will investigate generation of DC-shells without segmentation. In addition, as indicated at the end of Section 5.3.2, volume loss may be further reduced without SVM.



(a) $\mu = 0.01$ (b) $\mu = 0.1$ (c) $\mu = 1.0$ (d) $\mu = 10$

(e) $\mu = 0.01$ (f) $\mu = 0.1$ (g) $\mu = 1.0$ (h) $\mu = 10$

(i) $\mu = 100$ (j) $\mu = 1K$ (k) $\mu = 10K$

Figure 5.9: **top**: Changing penalty parameter μ from 0.01 to 10 affects SVM cuts. **bottom**: Gaps shown in red below the spikes narrow at different rates as μ increases from 0.1 to 10,000 because the spike on the left is buried deeper than the one on the right.

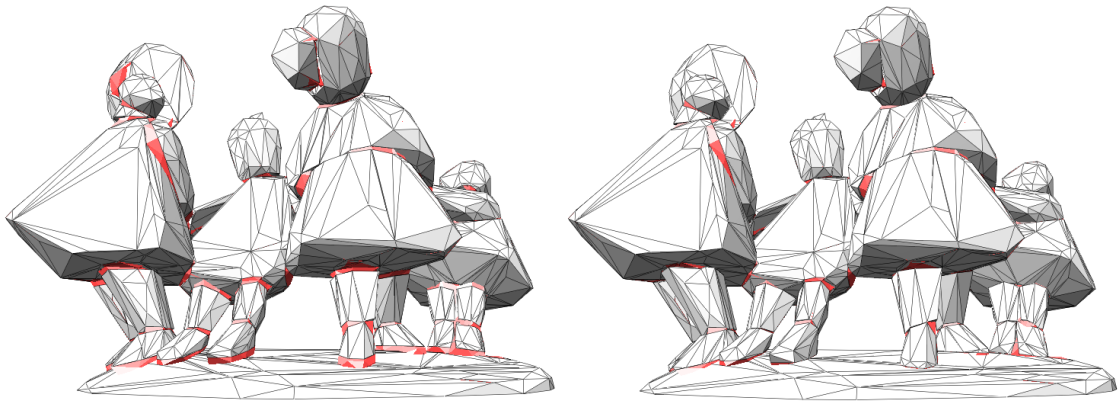


Figure 5.10: **Left**: DC-shells created using SVM. The red regions are volumes trimmed from the original input to create disjoint convex objects. The volume loss 1.62% of the volume of the union of the input convex hulls. **Right**: DC-shells created using exact volume computation. The volume loss is at 0.86%.

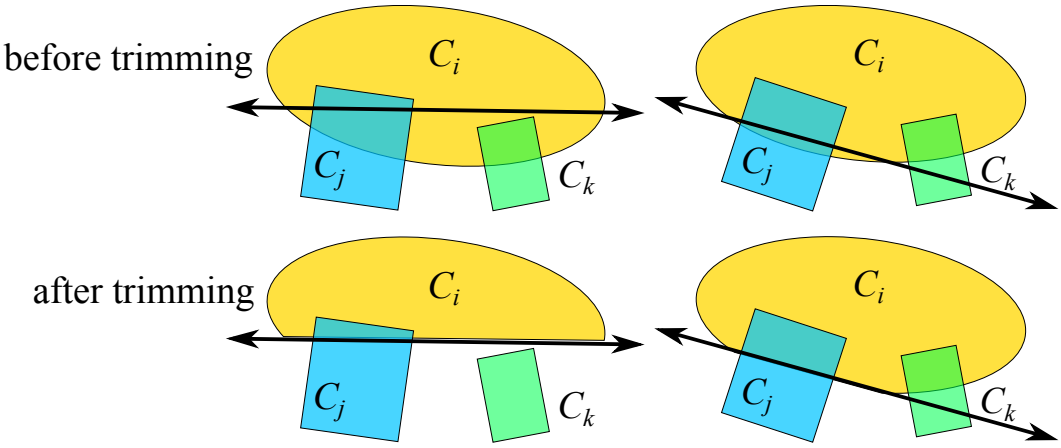


Figure 5.11: The cut between C_i and C_j on the left interferes with C_i and C_j. The cut on the right does not.

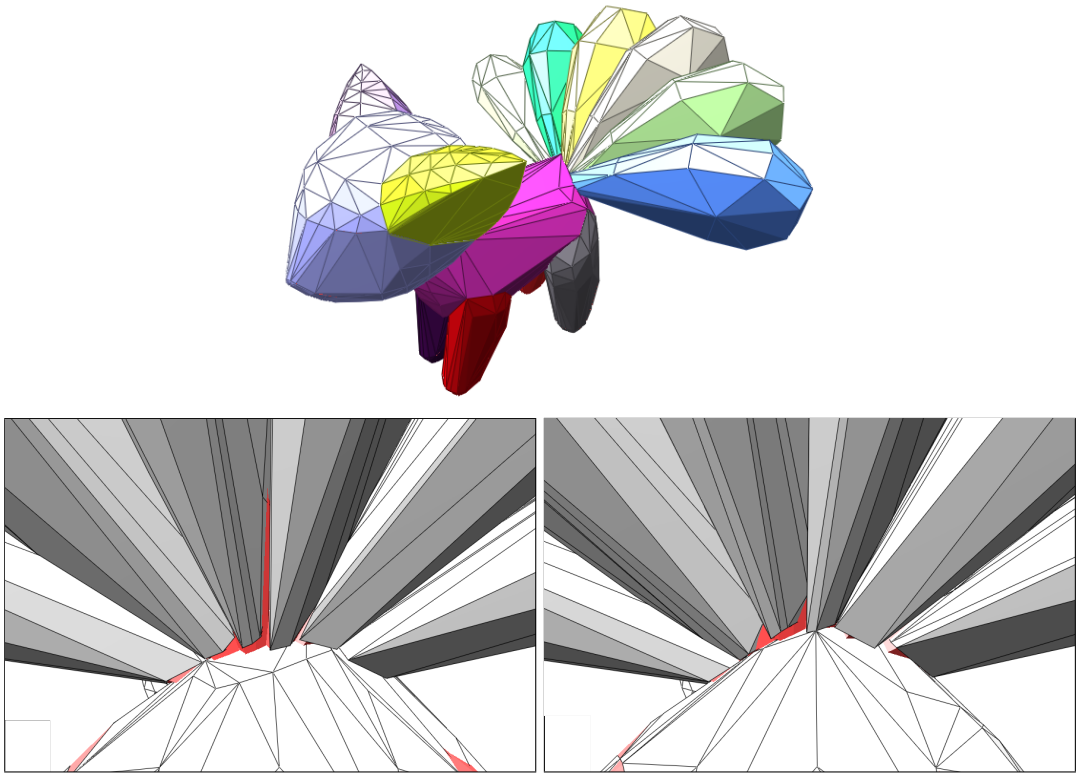
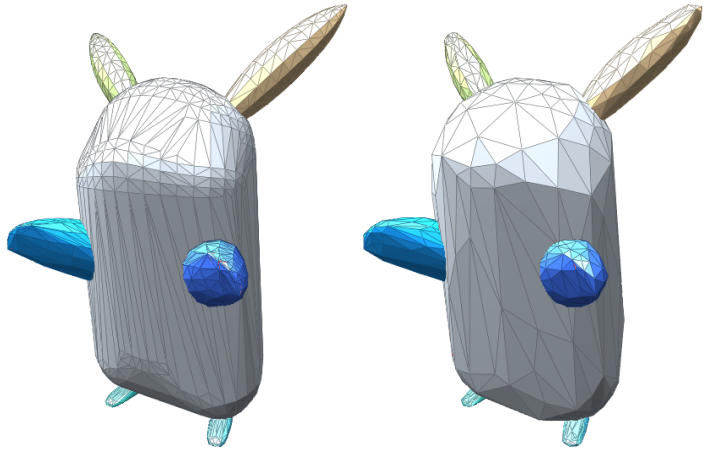


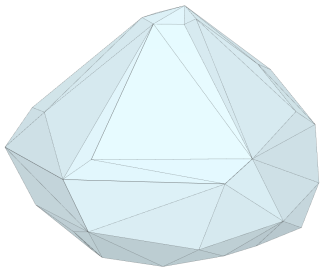
Figure 5.12: Cuts between the tails and the torso of a Vulpix model interference and make some tails completed separated from the torso.



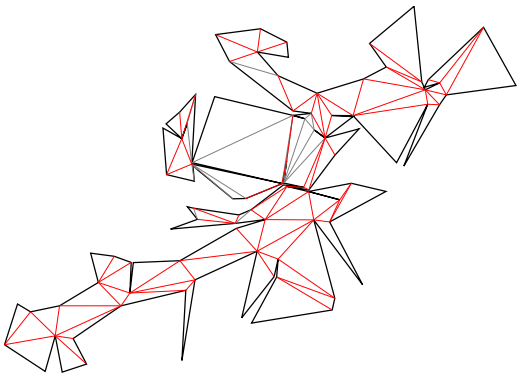
(a) original hull (b) our remesh

(c) isotropic remesh of (a) (d) convex hulls of (c)

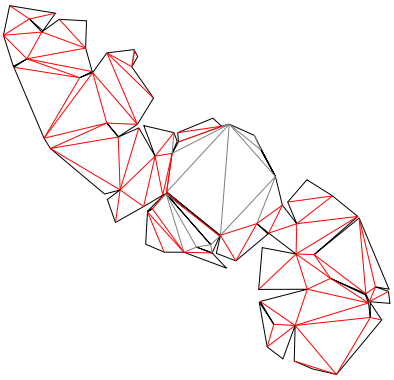
Figure 5.13: Remeshing results using the proposed method (b) and using isotropic remesh (c) and then reenforce convexity using convex hulls (d). Notice the skinny triangles on Pikachu's belly in (d).



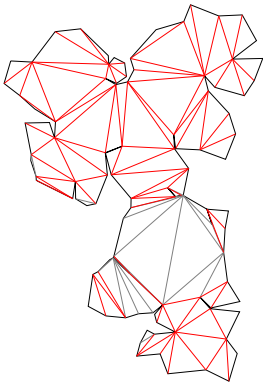
(a) Model



(b) Before optimization.



(c) Minimal convex hull area.



(d) Minimal total cut length.

Figure 5.14: Optimized nets of a Bulbasaur's seed model. Convex hull area is reduced by 41.42% and total cut length is reduced by 50.41%.

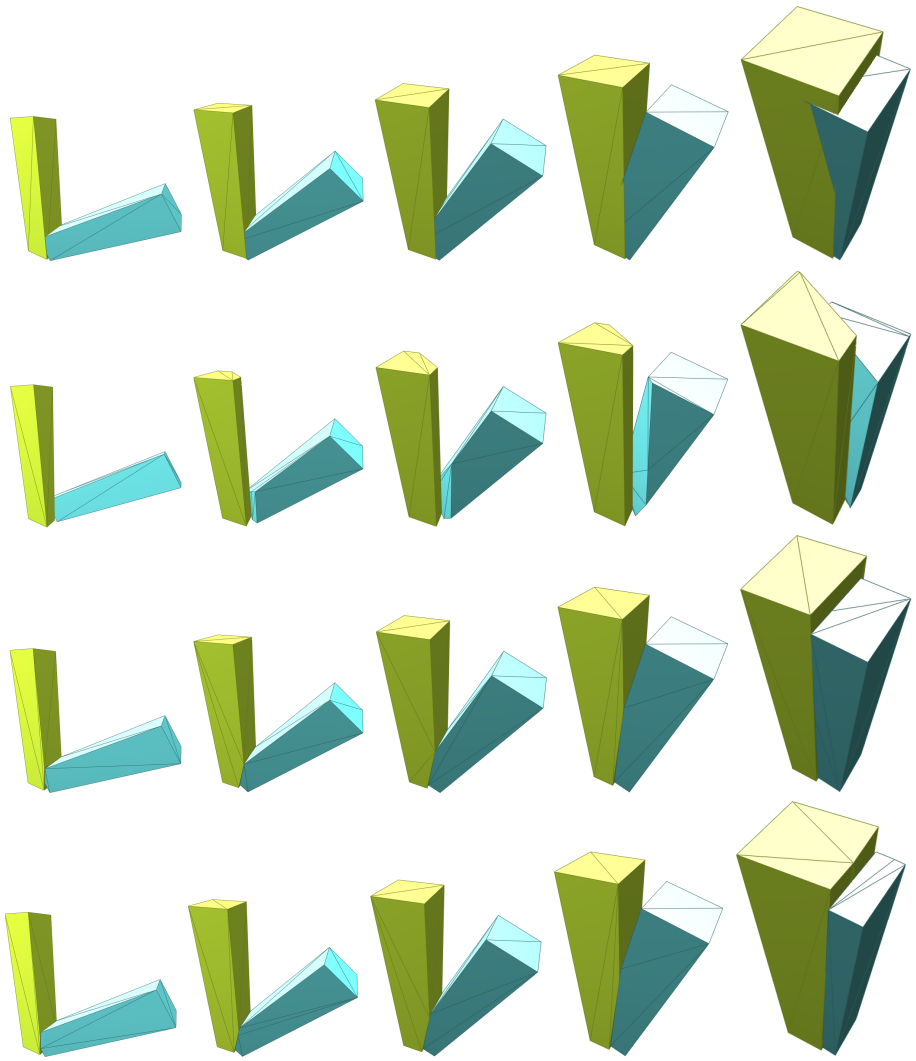
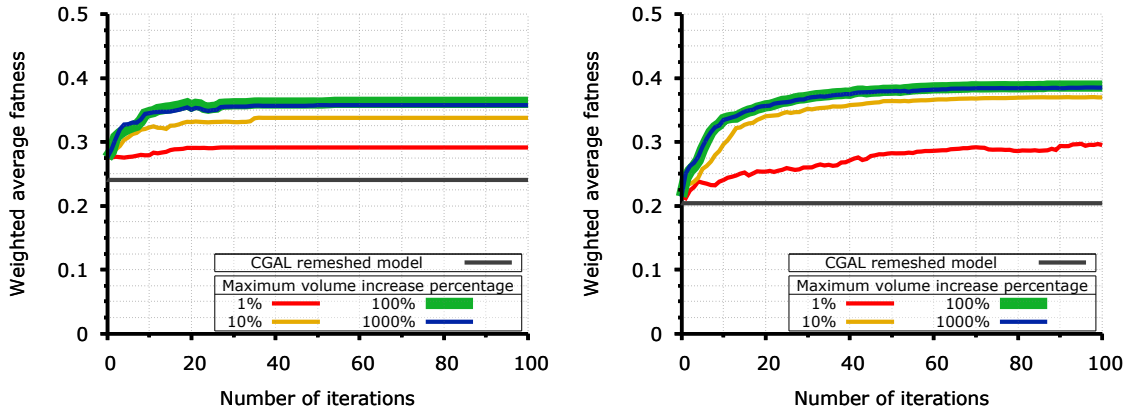
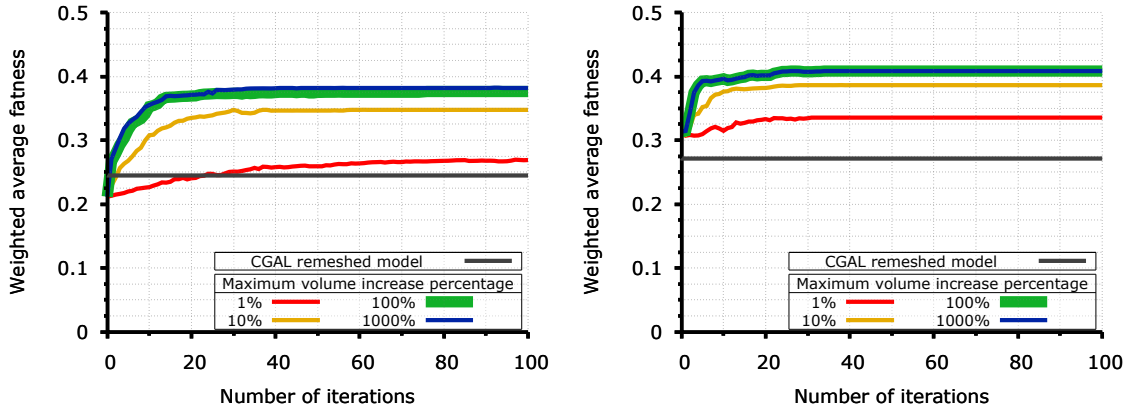


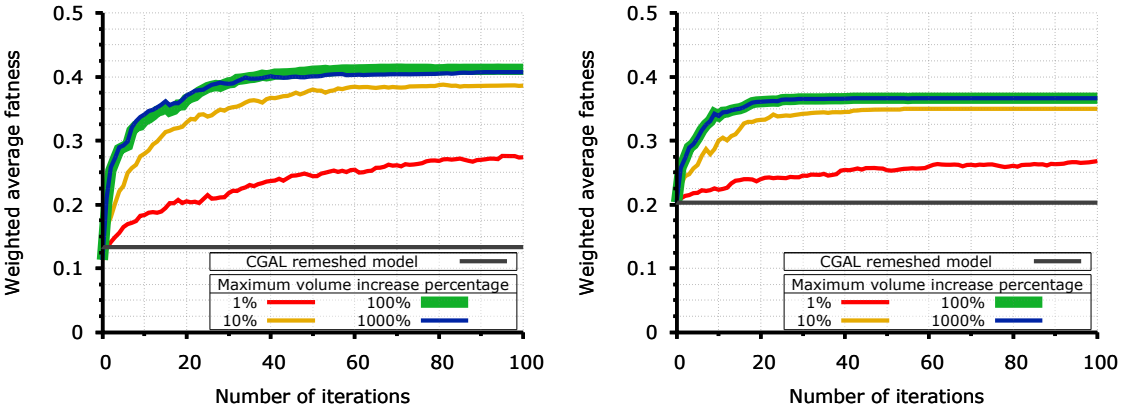
Figure 5.15: A controlled study with two intersecting bars. From top to bottom: input overlapping bars, and DC-shells created by least-squares fit heuristic, SVM and exact methods.



(a) Mother (b) Rocket

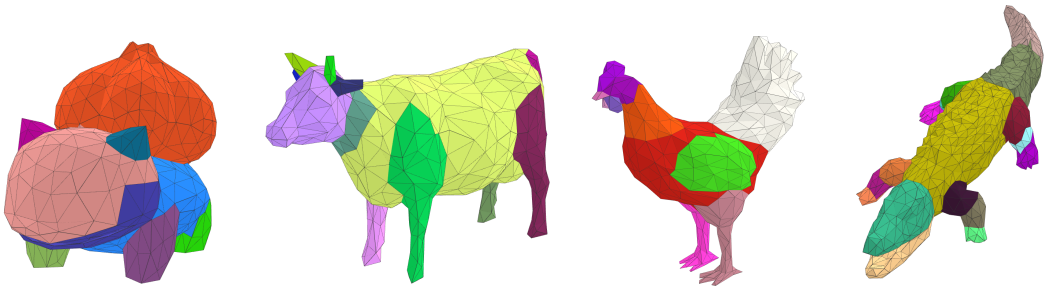


(c) Ultraman (d) Bulbasaur

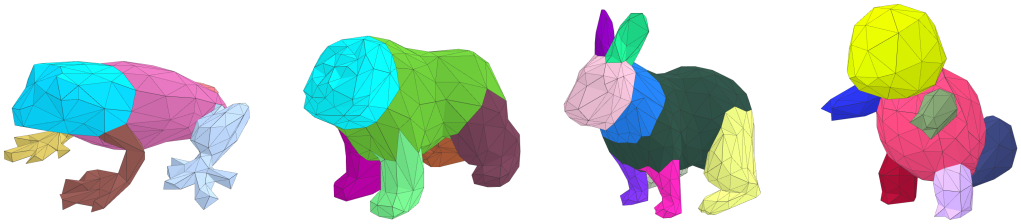


(e) Bull (f) Crocodile

Figure 5.16: Weighted average fatness with respect to number of iterations at four different percentages of maximum volume increase: 1%, 10%. 100%. and 1000%.



(a) Bulbasaur (b) Bull (c) Chicken (d) Crocodile



(e) Frog (f) Haechi (g) Rabbit (h) Squirtle

Figure 5.17: Models used in our fabrication experiments in Section 5.7.

Figure 5.18: Models built from the composite shape and DC shells.

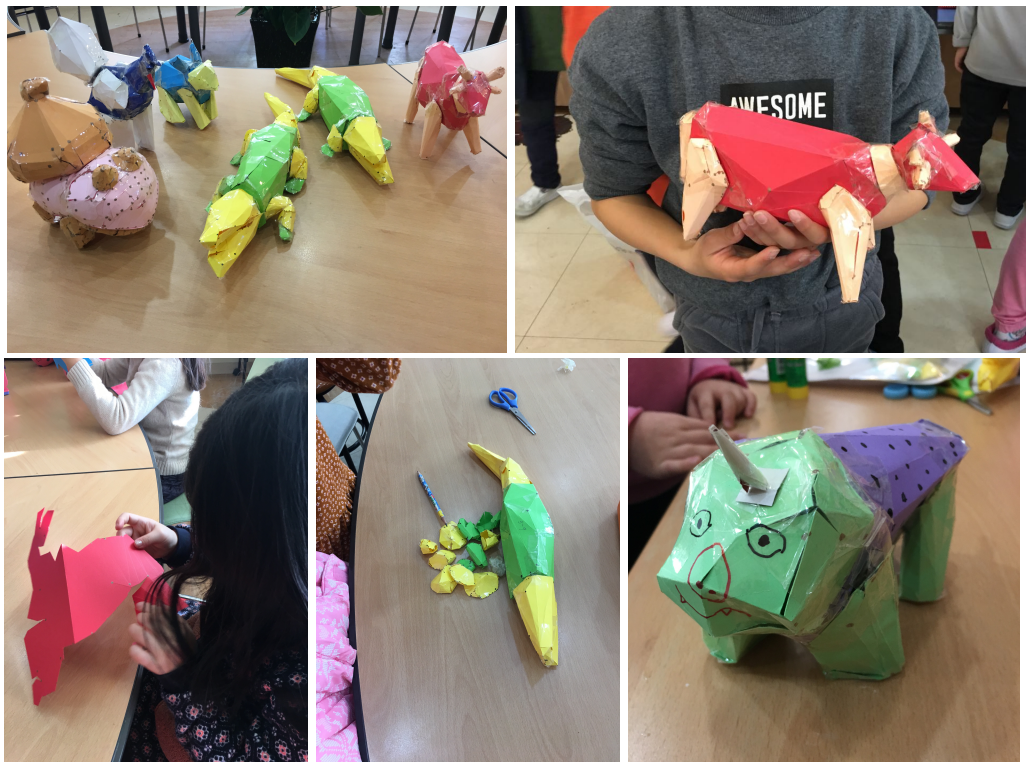


Figure 5.19: Paper crafts created by 9-year-old school children.

Chapter 6: Compact Folding

6.1 Introduction

Methods has been developed for simulating the folding process of a given crease pattern [9, 67] and for generating crease patterns from 3D shapes [18]. Most existing methods assume zero thickness material, a few new methods has been proposed in the literature to accommodate the thickness of the material with their own merits and demerits [68, 69, 70, 71].

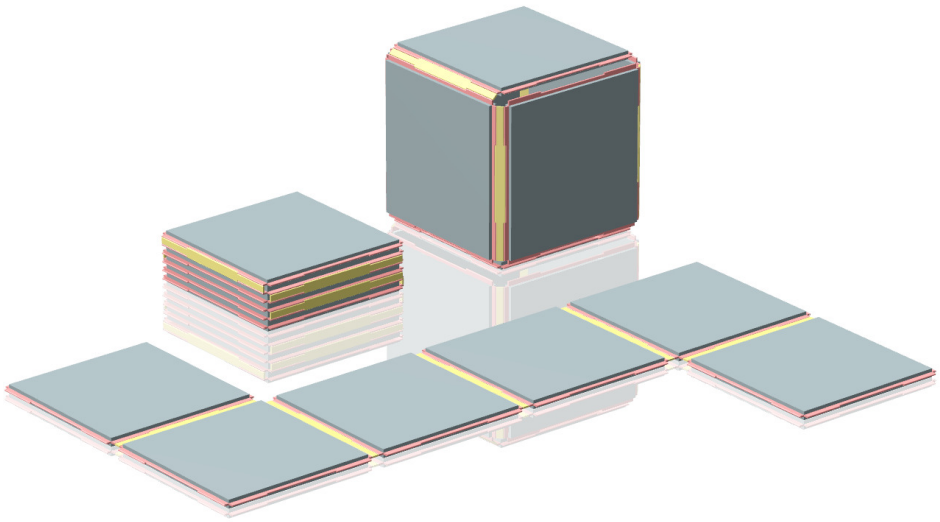
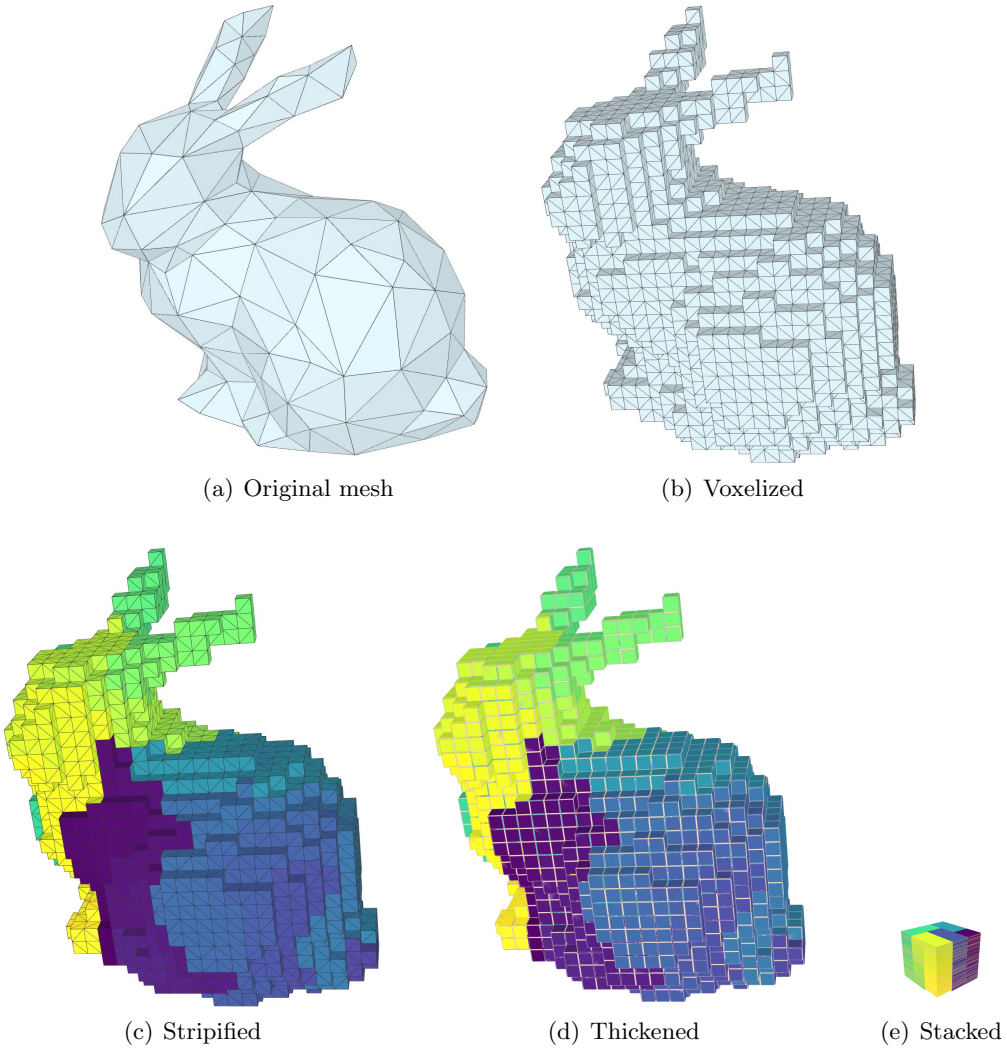


Figure 6.1: Comparison of the actual size of the folded, unfolded and stacked states of a cube model. The thickness of the panel is 5% of its size.

Assembling a polyhedron from one or multiple flat yet foldable/developable components is known as paper crafting which is another practical approach to fabricate complex 3D



(a) Original mesh (b) Voxelized

(c) Stripified (d) Thickened (e) Stacked

Figure 6.2: Pipeline of our approach.

shapes from flat materials [2, 12, 17, 19]. By cutting along a carefully selected subset of edges of the polyhedron, we could unfold the polyhedron into a planer shape on 2D space without overlapping which is called a net of the polyhedron. Both heuristic methods (mostly only work on convex shapes) and evolutionary algorithms are proposed Also, the huge dimension of the nets makes it hard to fabricate in practice.

One main advantage of foldable shapes is the compactness of their folded state as show in Fig. 6.1. This makes carrying of large objects, from maps, umbrellas, chairs, etc., used

in our daily life to the solar panels on the satellite, possible and much easier. [70] propose a method for designing origami-based deployable arrays with a high deployed-to-stowed ratio which can be used to design solar cells in the spacecraft. [72] present a method of transforming a 3D shape into a box.

In this disseration, we propose a new approach to fabricate foldable 3D shapes with limited working space by finding most compact folded states of the model. Unlike traditional methods that start from unfolded flat sheets and take the original model as the folded state, our method finds a Hamiltonian path in the mesh such that we can fold/stack all faces into one or multiple connected piles, the original model then can be obtained by unfold the piles. Experimental results show that our method can significantly reduce the workspace required to fabricate the models.

6.2 Related Works

6.2.1 Fold Thick Origamis

Mathematical models for folding rigid origami has been developed [67], however, assuming zero thickness material makes it hard to use in practice. Methods for accommodating thick material then were proposed in the literature, including: Axis-shift method [68], this method shifts the rotation axes to either top or bottom of the thick panel depends on the crease type (e.g. mountain or valley); Volume Trimming method [68], this method trims the edge of material to maintain the kinematics to a limited folding angle range; Offset Panel method [69], this method offsets the panels while maintain the rotation axes which can accommodate the full range of motion. Offset Crease method [70], this method widens creases with flexible material and add gaps for folds to accommodate thickness. A detailed comparison of these methods can be found in [71].

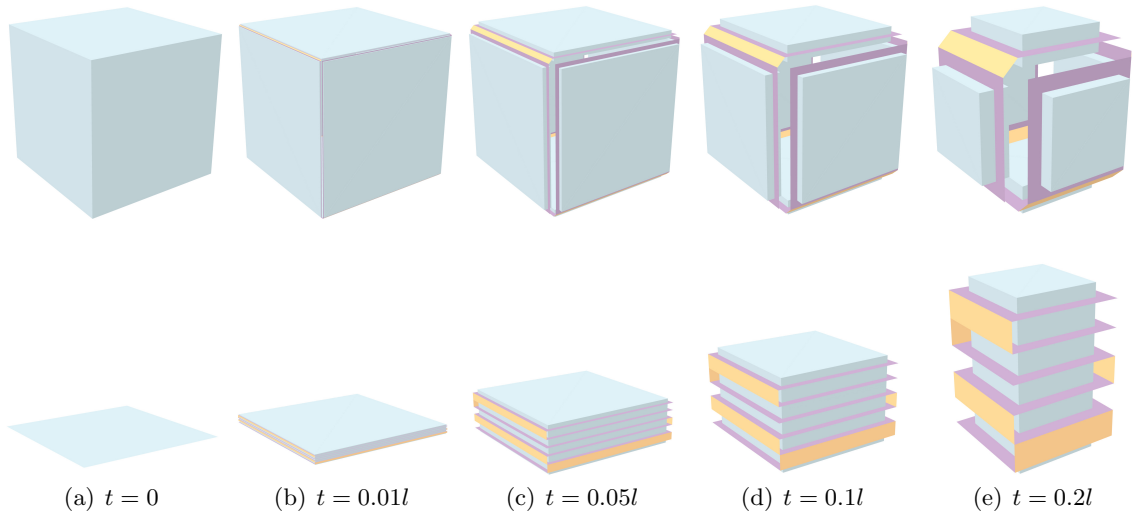
6.2.2 Mesh Stripification

The Hamiltonian path/cycle problems are to determine whether there exists a Hamiltonian path/cycle (a path/cycle in the graph that each vertex is visited exactly once) in the given graph which are both NP-complete. Finding a Hamiltonian path/cycle is as hard as determining its existence. The best algorithm so far finds a Hamiltonian cycle in $O(1.657^n)$ for a n-vertex graph and $O(1.251^n)$ for sparse graphs in which every node has a max degree of 3 [73]. Mesh stripification is a special case of the Hamiltonian path problem which has applications in computer graphics for fast rendering, mesh simplification and compression [74].

[75] finds a Hamiltonian triangulation of a quadrilateral mesh in linear time. It first finds cycles in the dual graph of the mesh and then merge the cycles by flipping the diagonal edges. Although this method is efficient, there is no direct extension to find a single quadrilateral strip. [76] uses the 2-factor partitioning of the dual graph of the quadrilateral mesh to find disjoint cycles and merge those cycles into one. However, this mesh requires a nontrivial refinement of the mesh to form the Hamiltonian cycle makes it not applicable to stacking problems.

6.3 Our Approach

We show the pipeline of our approach in Fig. 6.2. There are 4 main steps: mesh voxelization, mesh stripification, thickness accommodation and stacking. The main idea of our approach is to approximate the input mesh with equal size square panels. By finding a Hamiltonian path of the In Fig. 6.2(c), the faces are color-coded to represent the Hamiltonian path. The starting point is shown in purple, the middle point is shown in green and the end point is shown in yellow.



(a) $t = 0$ (b) $t = 0.01l$ (c) $t = 0.05l$ (d) $t = 0.1l$ (e) $t = 0.2l$

Figure 6.3: The folded state of a cube model and its corresponding stacked state under different thicknesses. l is the original panel size and t is the panel thickness.

6.3.1 Mesh Voxelization

Voxelization [77] is widely used in compute graphics with applications in visualization, fluid simulation, particle collision, etc. There exists two type of mesh voxelizations: surface voxelization and volume voxelization. In surface voexlization, only the surface region of model will be filled with the voxels while in volume voxelizaiton, voxels are also filled in the inner side of the mesh. We extract the out-most faces from the surface voxelization of the mesh which gives us a perfect mesh for stacking: all faces are identical squares. And each face can fold onto or under any one of its 4 neighboring faces.

6.3.2 Stripification of Quadrilateral Meshes

After voxelization, each face becomes a square and has exact 4 neighbor faces. We are guaranteed to find Hamiltonian paths in the dual graph of the voxelized mesh as a 4-regular graph such that we can cut the mesh and make it stackable.

Traveling Salesman Problem The exponential running time for finding Hamiltonian cycles prohibits its practical usage on quadrilateral meshes of thousands of faces. We convert

the Hamiltonian path problem (HPP) into the well known Traveling salesman problem (TSP) which finds a Hamiltonian cycle with minimum cost (the sum of crossed dual edge weights). We set the weight of a pair of nodes to 1 if there is an edge connects them and $+\infty$ otherwise, then we know the optimal solution is n if there are n faces in the original mesh.

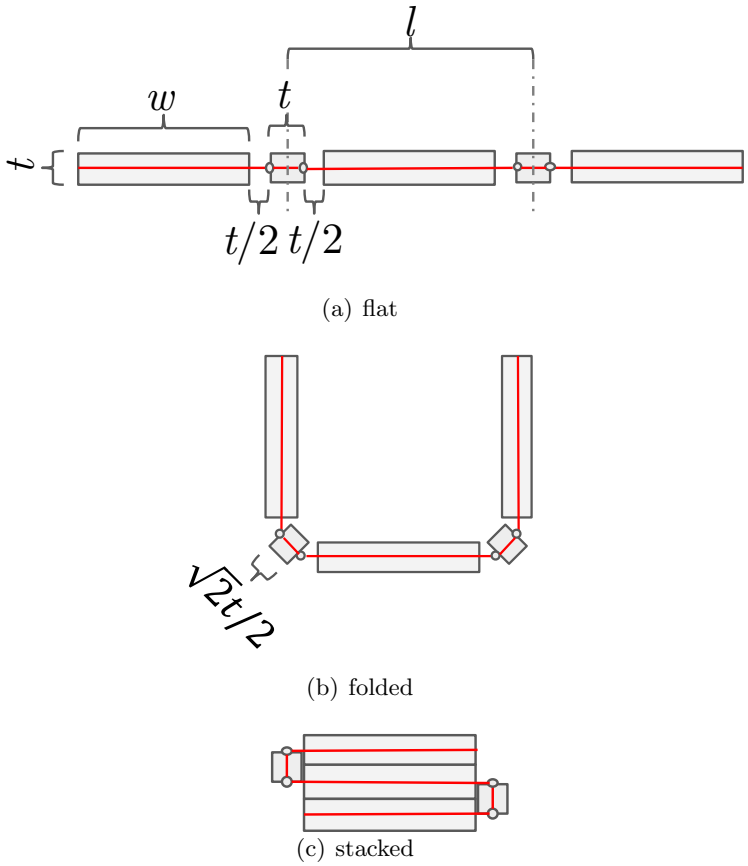
There are several reasons for us consider switching to TSP: 1) The solution of TSP can be converted to a set of solutions of HPP. 2) TSP is a well studied problem comparing to HPP, thus many solvers available in the public domain. 3) The pre-known upper bound n helps the solver to cut unnecessary branches thus find solutions more efficiently. We use public available yet state-of-art TSP solver Concorde [78] to find Hamiltonian paths in the dual graph of the mesh.

6.3.3 Thickness Accommodation

All previous methods for accommodating thickness have only one target state, the folded state. And a panel only folds in one direction, e.g. it is either a mountain fold or a valley fold. In this dissertation, we propose a new thickness accommodation method based on the Offset Crease method [70] which enables us fold the panels into two target states, the folded state and the stacked state. We will show that the proposed method guarantees that both target states are self-intersection free. We illustrate our proposed method in Fig. 6.4.

Panels Assuming the original (zero thickness) panel size is $l \times l$, panels of thickness t are trimmed to $(l-2t) \times (l-2t) \times t$ to accommodate thickness while enabling them to fold to both directions. Thus the maximum thickness that can be accommodated in our system is $t = 0.5l$.

Hinges The sliding hinge needs to have a variable length from $\frac{\sqrt{2}}{2}t$ to t during the entire folding process. We categorize the entire folding motion of a hinge into two stages: 1) *folding stage*. As shown in Fig. 6.5(a), $|\theta_i| \leq \frac{\pi}{2}$, we would like to preserve the kinematics as of folding zero thickness material such that we can ensure the final folded state is globally intersection free. 2) *stacking state*. As shown in Fig. 6.5(b), $|\theta_i| > \frac{\pi}{2}$, the goal is to smoothly



(a) flat

(b) folded

(c) stacked

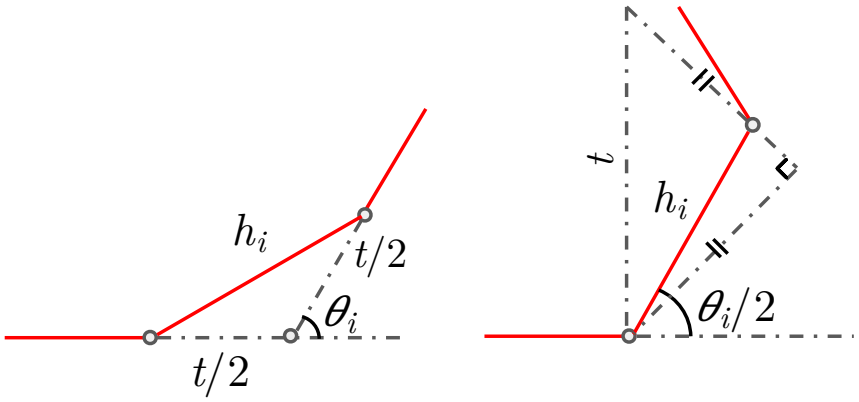
Figure 6.4: The proposed method to fold thick panels.

extend the hinge from $\frac{\sqrt{2}}{2}t$ to t to accommodate the thickness when stacked.

The length of the i-th hinge is derived in Eq. 6.1,

$$h_i = \begin{cases} cos(\frac{|\theta_i|}{2}) \cdot t, & |\theta_i| \leq \frac{\pi}{2} \\ \frac{\sqrt{2}}{2} \cdot sin(\frac{|\theta_i| - \frac{\pi}{2}}{2}) \cdot t, & |\theta_i| > \frac{\pi}{2} \end{cases} \tag{6.1}$$

where θ_i is the folding angle of the ideal crease, t is the thickness of the material. When $\theta_i = 0$, the flat state, $h_i = t$; $\theta_i = \pm\frac{\pi}{2}$, the maximum folded state, $h_i = \frac{\sqrt{2}}{2}t$; $\theta_i = \pm\pi$, the stacked state, $h_i = t$. During the entire folding range, we have $\frac{\sqrt{2}}{2}t \leq h_i \leq t$.



(a) folding stage: $|\theta_i| \leq \frac{\pi}{2}$ (b) stacking stage: $|\theta_i| > \frac{\pi}{2}$

Figure 6.5: Hinge length constraints during folding.

Connection We assume panels and hinges are connected by rigid thin material (shown as magenta panels in Fig. 6.3) whose size is $(l-t) \times (l-t) \times t_c$, $t_c \ll t$. The center of connection part is aligned with the center of the panel.

We simulate the folded and stacked states of a cube model with different thicknesses, the results are shown in Fig. 6.3.

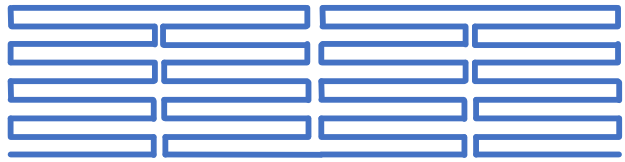
6.3.4 Stacking

Once we find a Hamiltonian cycle for a mesh, we can break the cycle at arbitrary position to get a strip. For a non-zero thickness panel in the strip, assuming only its neighboring panels can stack with it, one folded onto it and one folded under it. By assigning the folding angles of the panels properly along the path, we can stack all panels of the mesh into one or two piles.

Theorem 6.1. A single strip can be stacked into one or two piles.

Proof. Picking either end of the strip as the base face, a single pile stacking can be achieved by fold each child panel onto its parent panel along the strip. Unfolding the stacking at arbitrary position yields two piles. □

Type of Panels and Piles We say a pile is a *uphill* pile if the heights of its panels along the strip are increasing, otherwise it is a *downhill* pile. The *base panel* of a pile is the panel with height of 0. A penal is the *roof panel* of the pile if it is the highest one in that pile. The roof panel R_{up} of a uphill pile P_{up} can connect with the roof panel R_{down} of a downhill pile P_{down}, while the base panel B_{down} of a downhill pile P_{down} can connect with the base panel B'_{up} of another uphill pile P'_{up}. P_{up} and P_{down} must have the same heights in order to connect at the roof, while P_{down} and P'_{up} can have different heights since they connect at the base. The remaining questions is how to determine the heights for each pile. We discuss two assigning strategies, uniform stacking and non-uniform stacking as shown in Fig. 6.6, to stack the mesh into multiple piles (e.g. 2x2, 3x3), a more compact state, in details in the following.



(a) Uniform stacking

(b) Non-uniform stacking

Figure 6.6: Front view of two stacking strategies.

Uniform Stacking In uniform stacking, as shown in Fig. 6.6(a), all the piles have the same number of panels. Assuming we always start stacking with a uphill pile then an exception can be made for the last pile if it is a uphill pile. The last uphill pile can have different height, either higher or lower than the rest of the piles. Given a mesh with n

panels and the number of piles k, the height h of each pile can be $\lceil * \rceil\, n/k$.

Fold to Stacked State The folding angle of an edge that connects two faces is $\pi - \rho_i$, while ρ_i is the dihedral angle between two faces. Let θ_i and θ_i' denote the folding angle of edge e_i in the original mesh and in the stacked state respectively. Let P_i be the child panel of e_i, that is when e_i rotated, P_i will rotate with e_i.

$$\theta_i' = \begin{cases} 0, & P_i \text{ is a base or roof panel} \\ -\pi, & \text{Height of } P_i \text{ is odd} \\ \pi, & \text{Height of } P_i \text{ is even} \end{cases} \tag{6.2}$$

In order to fold the mesh into the stacked state, the angle needed to fold for edge e_i is $\theta_i' - \theta_i$. By assigning the folding angle for each edge based on the type of its child panel, we can fold the mesh into the stacked status in $O(n)$. However, not all stacked states are feasible since some piles might collide with others. By breaking the Hamiltonian cycle at different locations, we have up to $n-1$ different strips and stacked states. For each stacked state, we can check the coordinates of each pile in $O(k)$ to filter out invalid stackings. We show a mountain model and its stackings of different number of piles in Fig. 6.7.

Non-uniform Stacking We can relax the same height constraint when there is not enough variation to find a valid folded state via uniform stacking. Each pair of uphill and downhill piles still need to have the same height, while the downhill to uphill pair of piles can have different heights. For simplicity, the height of the later uphill pile is chosen from $\{h, h \pm l\}$, where l can be $1, 2, \cdots, m$, $m < h$. This gives us $m \cdot (3^{\lfloor * \rfloor k/2} - 1)$ different stacked states for each strip.

Most Compact Stacking For a given voxelized mesh m, its most compact stacking s can be computed based on the size and the thickness of the panel. The compactness C is measured as:

$$C = \frac{|W_s| + |D_s| + |H_s|}{|W_m| + |D_m| + |H_m|}, \tag{6.3}$$

where W, D, H represents the width, depth and height.

$H_s = t \lceil * \rceil \frac{|F|}{W_s D_s}$, Let $W_s = D_s$, then the optimal compactness $C = \frac{3\sqrt[3]{t|F|}}{|W_m|+|D_m|+|H_m|}$ can be obtained when $W_s = \sqrt[3]{t|F|}$, where $|F|$ is the number of faces in m and t is the thickness of the panel which has a dimension of 1×1.

We can also use the volume ratio to measure the compactness:

$$\text{Volume Ratio} = \frac{V_{bbox}}{V_{mesh}}, \tag{6.4}$$

which is simply the volume of bounding box of the stacking divided by the volume of the original mesh.

6.4 Experimental Results

6.4.1 Experimental Setup

We implemented the proposed method in C++. All data are collected on a Macbook Pro with a 2.5 GHz Intel Core i7 CPU with 16GB Memory running macOS 10.12. We show the models used in the experiments in Fig. 6.8.

6.4.2 Finding Hamiltonian Paths

We use Concorde TSP solver to find Hamilton paths in the dual graph of the mesh. The running time for finding a Hamilton path on different models in Table 6.1 from which we can see that the running times grow almost linearly to the number of quads in the mesh.

6.4.3 Most Compact Stacking

We show the most compact stackings of the Bunny model shown in Fig. 6.8(d) under different thickness in Fig. 6.9. The compactness and volume ratio of the optimal stackings of the Bunny model is listed in Table 6.2.

6.4.4 Finding Continous Folding Motions

We use motion planning technique to find continuous folding motion between the stacked state the target state. A discrete domain sampling based planner [11] is employed to find such motion to ensure it is physically realizable. We show the folding motion of the Mountain model (shown in Fig. 6.7) from 1-pile stacking to unfolded shape in Fig. 6.10. It takes, average of 30 trails, 79.09 seconds for the planner to find a continuous folding path for the chain.

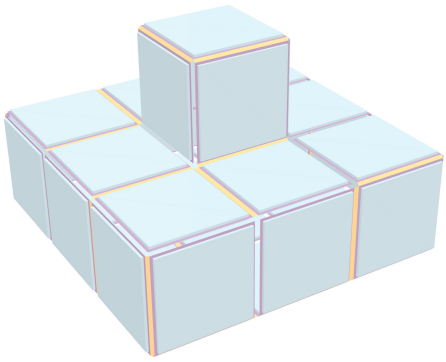
6.4.5 Physical Models

We show physical realization of the cube model using Lego to illustrate the proposed idea of two foldable target shapes. The panel size is 3.2in $\times$ 3.2in $\times$ 1.0in, the size of ideal zero thickness panel will be 3.6in $\times$ 3.6in. The net, the folded state and the stacked state of the cube model are shown in Fig. 6.11. We also show two others shapes that can foldable from the cube chain.

6.5 Conclusion

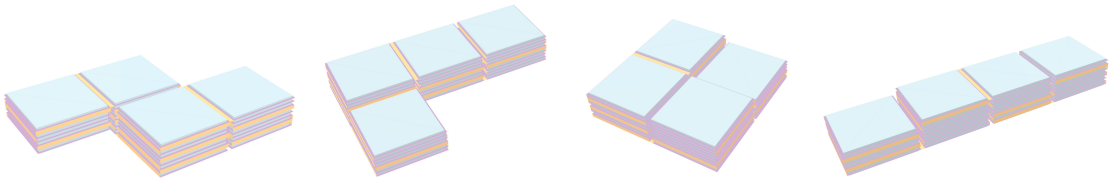
In this disseration, we propose a novel approach to fold a voxelized mesh into a much more compact form called stacking which enable us to fabricate (e.g. 3D-printing and unfolding) a large 3D model from a much smaller workspace. We show a technique to accommodate the thickness of the material which also enables the folding motion in both directions.

Limitations and Future Works In this disseration, we did not plan the folding motion for those complex thick chains by assuming there always exists a collision folding path due to its huge degree of freedom (DOF). Meanwhile, the high degree of freedom (DOF) makes the thick chains hard to fold for both humans and themselves as self-folding machines. We are seeking other representations instead of voxelization to obtain a better yet stackable approximation of the original model.

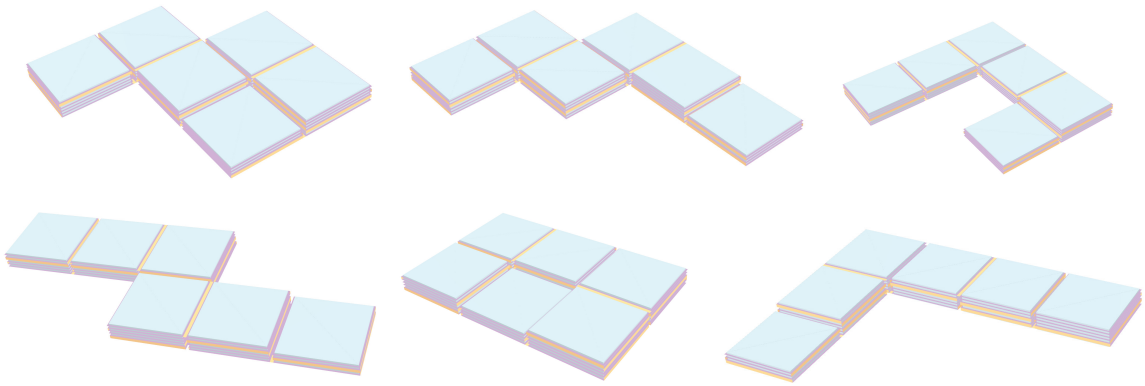


(a) The mountain model

(b) 1-pile stacking.



(c) Stackings of 4 piles.

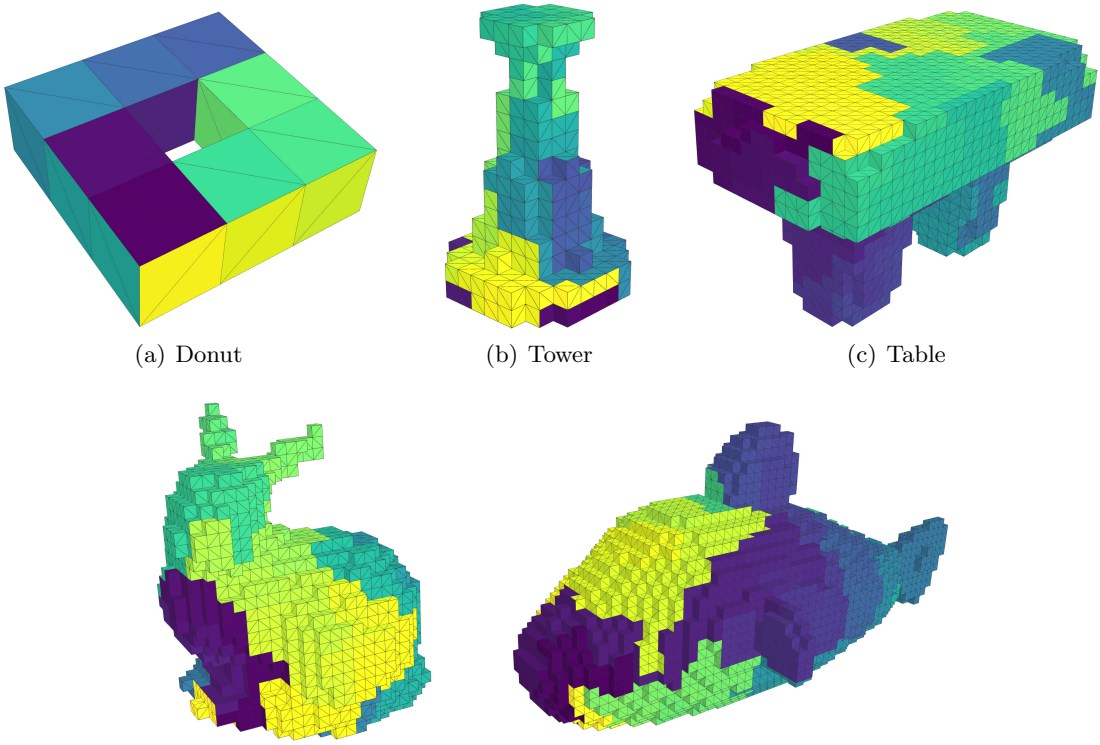


(d) Stackings of 6 piles.

Figure 6.7: A mountain model and its representative stackings of different number of piles.

Model	# of Quads	Time (s)
Donut	32	0.01
Tower	462	0.59
Table	1964	3.68
Bunny	2206	2.50
Fish	4396	5.41

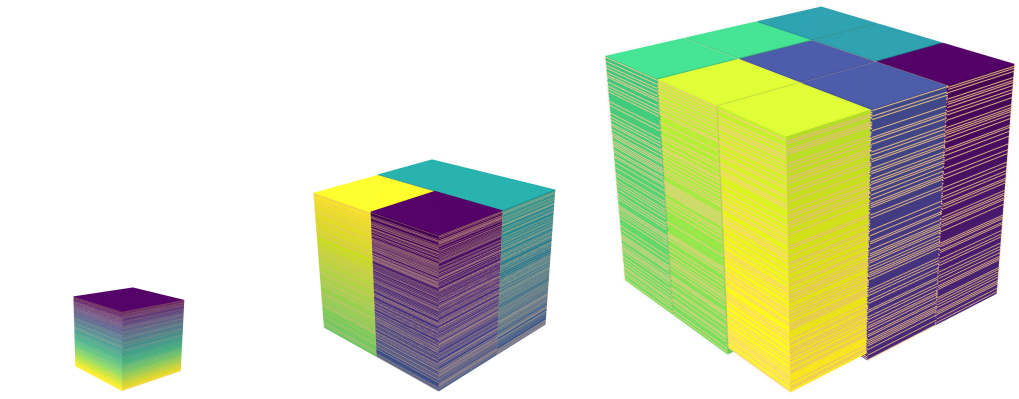
Table 6.1: Running time of finding a Hamiltonian path.



(a) Donut (b) Tower (c) Table

(d) Bunny (e) Fish

Figure 6.8: The color coded Hamiltonian paths of the models.



(a) $1 \times 1, t = 5 \times 10^{-4}$ (b) $2 \times 2, t = 4 \times 10^{-3}$ (c) $3 \times 3, t = 1.35 \times 10^{-2}$

Figure 6.9: The compactest stacked states of the Bunny model (Fig. 6.8(d)) under different thicknesses.

t	Base Size	Compactness	Volume Ratio
$0.0001 \times l$	1×1	0.0305	5.7776×10^{-5}
$0.001 \times l$	2×2	0.0632	5.7776×10^{-4}
$0.01 \times l$	4×4	0.1173	5.7776×10^{-3}
$0.1 \times l$	8×8	0.3024	5.7776×10^{-2}
$0.2 \times l$	10×10	0.3780	1.1284×10^{-1}

Table 6.2: The optimal compactness and volume ratio of the stacked Bunny model under different thicknesses.

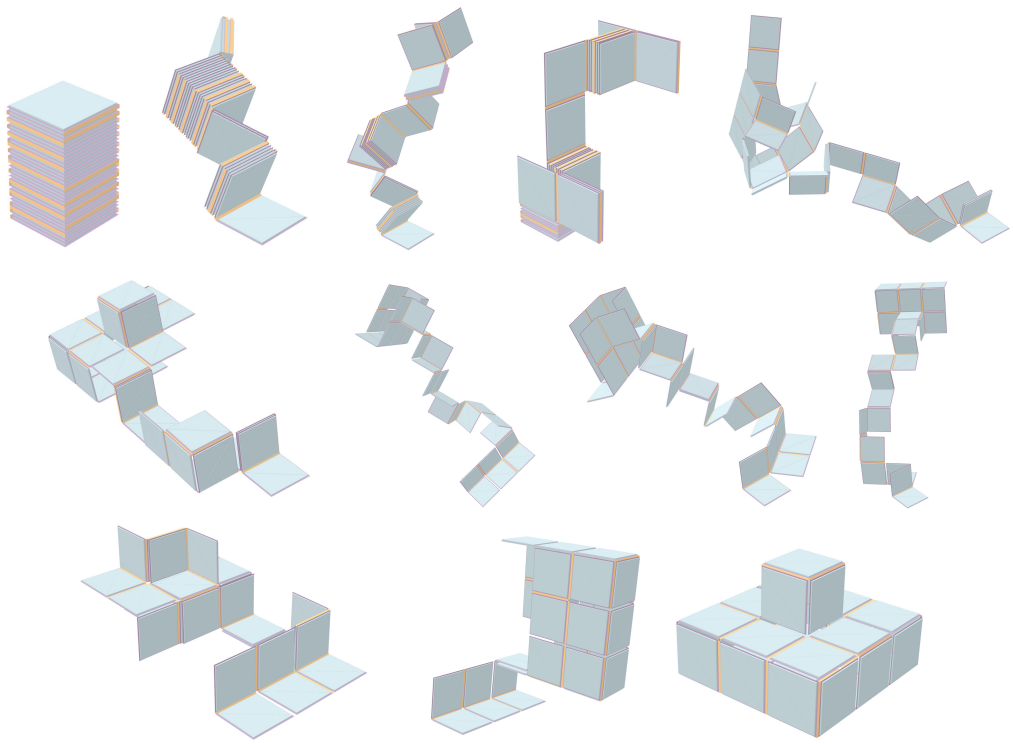
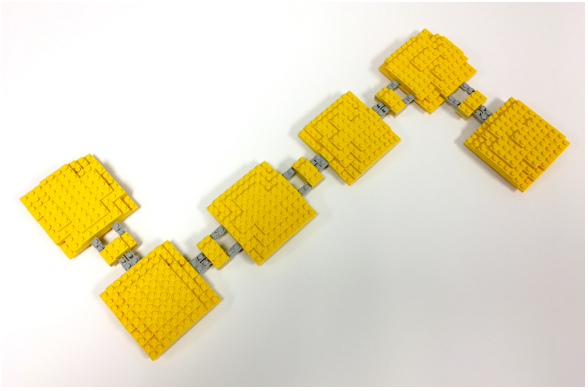
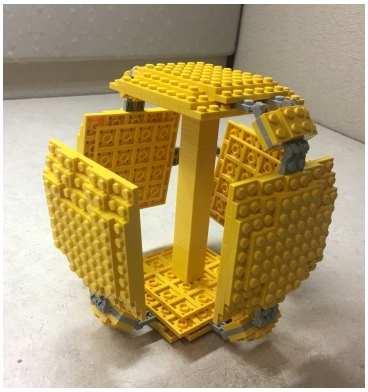


Figure 6.10: The continuous unfolding motion of the Mountain model from its stacked shape to target shape. The folding motion can be best visualized using our web-based interactive folder at https://goo.gl/BDSWbd.

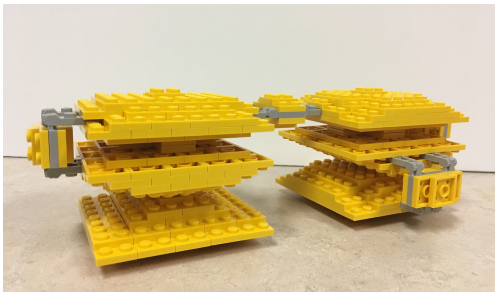


(a) Net

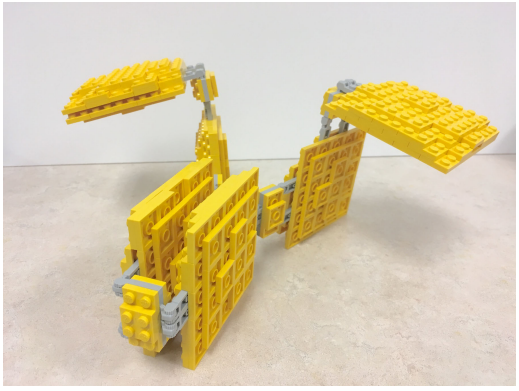


(b) Folded

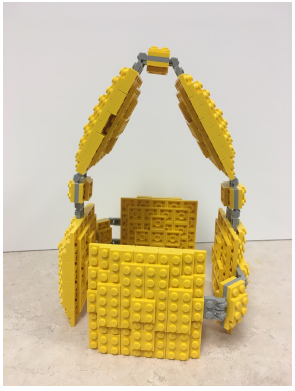
(c) Stacked



(d) Stacked (2 piles)



(e) Plane



(f) House

Figure 6.11: A Lego realization of the cube model and two other shapes folded from the thick panel chain.

Chapter 7: Conclusions

The main thread of this dissertation was to make shapes foldable. We discussed the modeling two types of foldable objects: rigid origami and unfoldings of polyhedra. For edge unfoldings, a genetic algorithm was proposed to find non-overlapping unfoldings for non-convex shapes. Based on that, we propose a learning-based approach to simultaneously segment and unfold a complex non-convex shape. We also present a novel approach to fold complex 3D shapes into its compactest states via stacking. For both types of foldable objects, motion planning technique was used to find a feasible continuous folding path which is key step towards building self-folding machines. We present a new sampling strategy to sample in the discrete domain which significantly increase the probability of generating valid samples thus finds feasible path more efficiently.

7.1 Future Researches

Although our preliminary results are promising, the work in this dissertation still has many concerns and several problems remain open. The shape and topology of nets play a very important role in planning continuous folding motion, e.g., for a given shape, some nets of it are easier to fold than others in terms of path planning time. We would like to find/generate optimal nets for a given mesh. More specifically, we are looking for nets that are (nearly) linearly foldable. We say a net is linearly foldable if it has a straight line path in $\mathcal{C}_{free}$, whose path planning time is negligible comparing to non-linearly foldable nets. Most importantly, linearly foldable nets are more suitable for self-folding robots whose actuation is hard or impossible to control (e.g. via uniform heating) since folding all the creases at the same speed transforms the net to the target shape continuously without self-intersection.

Bibliography

[1] L. Glover, "KitRex," 2014. [Online]. Available: http://www.kit-rex.com/

[2] S. Takahashi, H.-Y. Wu, S. H. Saw, C.-C. Lin, and H.-C. Yen, "Optimized topological surgery for unfolding 3d meshes," in *Computer Graphics Forum*, vol. 30, no. 7. Wiley Online Library, 2011, pp. 2077–2086.

[3] W. Schlickenrieder, "Nets of polyhedra," Master's thesis, Technische Universität Berlin, 1997.

[4] S. Felton, M. Tolley, E. Demaine, D. Rus, and R. Wood, "A method for building self-folding machines," *Science*, vol. 345, no. 6197, pp. 644–646, 2014.

[5] B. An, N. Benbernou, E. D. Demaine, and D. Rus, "Planning to fold multiple objects from a single self-folding sheet," *Robotica*, vol. 29, no. 1, pp. 87–102, 2011.

[6] L. Swanstrom, M. Whiteford, and Y. Khajanchee, "Developing essential tools to enable transgastric surgery," *Surgical endoscopy*, vol. 22, no. 3, pp. 600–604, 2008.

[7] T. Tachi, "Designing freeform origami tessellations by generalizing resch's patterns," *Journal of Mechanical Design*, vol. 135, no. 11, p. 111006, 2013.

[8] G. Song and N. M. Amato, "A motion-planning approach to folding: From paper craft to protein folding," *Robotics and Automation, IEEE Transactions on*, vol. 20, no. 1, pp. 60–71, 2004.

[9] Z. Xi and J.-M. Lien, "Folding rigid origami with closure constraints," in *International Design and Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE)*. Buffalo, NY: ASME, Aug. 2014.

[10] D. Dougherty, "The maker movement," *innovations*, vol. 7, no. 3, pp. 11–14, 2012.

[11] Z. Xi and J.-M. Lien, "Continuous unfolding of polyhedra - a motion planning approach," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, Sep. 2015, pp. 3249 – 3254.

[12] I. Shatz, A. Tal, and G. Leifman, "Paper craft models from meshes," *The Visual Computer*, vol. 22, no. 9-11, pp. 825–834, 2006.

[13] F. Massarwi, C. Gotsman, and G. Elber, "Papercraft models using generalized cylinders," in *Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on.* IEEE, 2007, pp. 148–157.

[14] D. Julius, V. Kraevoy, and A. Sheffer, "D-charts: Quasi-developable mesh segmentation," in *Computer Graphics Forum*, vol. 24, no. 3. Wiley Online Library, 2005, pp. 581–590.

[15] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 954–961, 2003.

[16] R. Straub and H. Prautzsch, "Creating optimized cut-out sheets for paper models from meshes," Karlsruhe Institute of Technology, Tech. Rep., 2011.

[17] J. Mitani and H. Suzuki, "Making papercraft toys from meshes using strip-based approximate unfolding," in *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3. ACM, 2004, pp. 259–263.

[18] T. Tachi, "Origamizing polyhedral surfaces," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 16, no. 2, pp. 298–311, 2010.

[19] Z. Xi, Y. hyeong Kim, Y. J. Kim, and J.-M. Lien, "Learning to segment and unfold polyhedral mesh from failures," in *Shape Modeling International (SMI); also appears in Journal of Computers & Graphics*, Berlin, Germany, Jun. 2016.

[20] S. Ahmed, C. Lauff, A. Crivaro, K. McGough, R. Sheridan, M. Frecker, P. von Lockette, Z. Ounaies, T. Simpson, J.-M. Lien, and R. Strzelec, "Multi-field responsive origami structures: Preliminary modeling and experiments," in *Proceedings of the ASME 2013 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, August 2013.

[21] S.-M. Belcastro and T. Hull, "A mathematical model for non-flat origami," in *Origami3: Proc. the 3rd International Meeting of Origami Mathematics, Science, and Education*, 2002, pp. 39–51.

[22] S. Miyazaki, T. Yasuda, S. Yokoi, and J.-i. Toriwaki, "An origami playing simulator in the virtual space," *Journal of Visualization and Computer Animation*, vol. 7, no. 1, pp. 25–42, 1996.

[23] D. J. Balkcom and M. T. Mason, "Robotic origami folding," *The International Journal of Robotics Research*, vol. 27, no. 5, pp. 613–627, 2008.

[24] T. Tachi, "Simulation of rigid origami," in *Origami4: Proceedings of The Fourth International Conference on Origami in Science, Mathematics, and Education*, 2009.

[25] J. H. Yakey, S. M. LaValle, and L. E. Kavraki, "Randomized path planning for linkages with closed kinematic chains," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 951–958, 2001.

[26] J. Cortes, T. Simeon, and J. P. Laumond, "A random loop generator for planning the motions of closed kinematic chains using PRM methods," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2002, pp. 2141–2146.

[27] L. Han and N. M. Amato, "A kinematics-based probabilistic roadmap method for closed chain systems," in *Robotics:New Directions*. Natick, MA: A K Peters, 2000, pp. 233–246, book containts the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Dartmouth, March 2000.

[28] J. Cortes and T. Simeon, "Sampling-based motion planning under kinematic loop-closure constraints," in *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*, 2004, to appear.

[29] D. Xie and N. M. Amato, "A kinematics-based probabilistic roadmap method for high dof closed chain systems," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 1. IEEE, 2004, pp. 473–478.

[30] X. Tang, S. Thomas, P. Coleman, and N. M. Amato, "Reachable distance space: Efficient sampling-based planning for spatially constrained systems," *The international journal of robotics research*, vol. 29, no. 7, pp. 916–934, 2010.

[31] J. O'Rourke, "Unfolding polyhedra," 2008.

[32] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polyhedra," in *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*. New York, NY, USA: ACM Press, 2007, pp. 121–131. [Online]. Available: http://doi.acm.org/10.1145/1236246.1236265

[33] S. Asafi, A. Goren, and D. Cohen-Or, "Weak convex decomposition by lines-of-sight," in *Computer Graphics Forum*, vol. 32, no. 5. Wiley Online Library, 2013, pp. 23–31.

[34] G. Liu, Z. Xi, and J.-M. Lien, "Nearly convex segmentation of polyhedra through convex ridge separation," in *Symposium on Solid & Physical Modeling (SPM); also appears in Journal of Computer-Aided Design*, Berlin, Germany, Jun. 2016.

[35] K. MIURA, "Proposition of pseudo-cylindrical concave polyhedral shells," *ISAS report*, vol. 34, no. 9, pp. 141–163, 1969.

[36] Z. Xi and J.-M. Lien, "Folding and unfolding origami tessellation by reusing folding path," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May. 2015.

[37] R. Bohlin and E. Kavraki, "Path planning using lazy prm," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1. IEEE, 2000, pp. 521–528.

[38] J.-M. Lien and Y. Lu, "Planning motion in similar environments," in *Proceedings of the Robotics: Science and Systems Conference (RSS)*, Seattle, Washington, Jun 2009.

[39] B. An, S. Miyashita, M. T. Tolley, D. M. Aukes, L. Meeker, E. D. Demaine, M. L. Demaine, R. J. Wood, and D. Rus, "An end-to-end approach to making self-folded 3d surface shapes by uniform heating," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1466–1473.

[40] Y. Liu, J. K. Boyles, J. Genzer, and M. D. Dickey, "Self-folding of polymer sheets using local light absorption," *Soft Matter*, vol. 8, no. 6, pp. 1764–1769, 2012.

[41] G. Liu, Y. Gingold, and J.-M. Lien, "Continuous visibility feature," in *28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA: IEEE, Jun. 2015, pp. 1182 – 1190.

[42] E. D. Demaine, M. L. Demaine, V. Hart, J. Iacono, S. Langerman, and J. O'Rourke, "Continuous blooming of convex polyhedra," *Graphs and Combinatorics*, vol. 27, no. 3, pp. 363–376, 2011.

[43] B. Chazelle, "Convex decompositions of polyhedra," in *Proc. 13th Annu. ACM Sympos. Theory Comput.*, 1981, pp. 70–79.

[44] M. Ghomi, "Affine unfoldings of convex polyhedra," *Geometry & Topology*, vol. 18, no. 5, pp. 3055–3090, 2014.

[45] H. Edelsbrunner, A. D. Robison, and X. Shen, "Covering convex sets with non-overlapping polygons," *Discrete Math.*, vol. 81, pp. 153–164, 1990.

[46] C. Bajaj and T. K. Dey, "Convex decomposition of polyhedra and robustness," *SIAM J. Comput.*, vol. 21, pp. 339–364, 1992.

[47] B. Chazelle, D. Dobkin, N. Shouraboura, and A. Tal, "Strategies for polyhedral surface decomposition: An experimental study," *Comput. Geom. Theory Appl.*, vol. 7, pp. 327–342, 1997.

[48] K. Mamou and F. Ghorbel, "A simple and efficient approach for 3d mesh approximate convex decomposition," in *Image Processing (ICIP), 2009 16th IEEE International Conference on.* IEEE, 2009, pp. 3501–3504.

[49] M. Ghosh, N. M. Amato, Y. Lu, and J.-M. Lien, "Fast approximate convex decomposition using relative concavity," *Computer-Aided Design*, vol. 45, no. 2, pp. 494–504, 2013.

[50] O. van Kaick, N. Fish, Y. Kleiman, S. Asafi, and D. Cohen-Or, "Shape segmentation by approximate convexity analysis," *ACM Trans. on Graphics*, vol. to appear, 2014.

[51] Z. Xi and J.-M. Lien, "Plan folding motion for rigid origami via discrete domain sampling," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May. 2015.

[52] L. Shapira, A. Shamir, and D. Cohen-Or, "Consistent mesh partitioning and skeletonisation using the shape diameter function," *The Visual Computer*, vol. 24, no. 4, pp. 249–259, 2008.

[53] J. Lawrence, "Polytope volume computation," *Math. Comput.*, vol. 57, no. 195, pp. 259–271, 1991.

[54] B. Büeler, A. Enge, K. Fukuda, and H.-J. Lüthi, "Exact volume computations for polytopes: a practical study," in *Abstracts 12th European Workshop Comput. Geom.* Universität Münster, 1996, pp. 57–64.

[55] L. Khachiyan, "Complexity of polytope volume computation," in *New Trends in Discrete and Computational Geometry*, ser. Algorithms and Combinatorics, J. Pach, Ed. Springer-Verlag, 1993, vol. 10, pp. 91–101.

[56] I. Z. Emiris and V. Fisikopoulos, "Efficient random-walk methods for approximating polytope volume," in *Proceedings of the thirtieth annual symposium on Computational geometry*. ACM, 2014, p. 318.

[57] R. L. Smith, "Efficient monte carlo procedures for generating points uniformly distributed over bounded regions," *Operations Research*, vol. 32, no. 6, pp. 1296–1308, 1984.

[58] H. Edelsbrunner and R. Waupotitsch, "Computing a ham-sandwich cut in two dimensions," *J. Symbolic Comput.*, vol. 2, pp. 171–178, 1986.

[59] C.-Y. Lo and W. Steiger, "An optimal-time algorithm for ham-sandwich cuts in the plane," in *Proc. 2nd Canad. Conf. Comput. Geom.*, 1990, pp. 5–9.

[60] S. Bespamyatnikh, D. Kirkpatrick, and J. Snoeyink, "Generalizing ham sandwich cuts to equitable subdivisions," *Discrete Comput. Geom.*, vol. 24, no. 4, pp. 605–622, 2000.

[61] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[62] M. Botsch, *Polygon mesh processing*. Natick, Mass: A K Peters, 2010.

[63] P. V. Sander, X. Gu, S. J. Gortler, H. Hoppe, and J. Snyder, "Silhouette clipping," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 327–334.

[64] P. Alliez, E. De Verdire, O. Devillers, and M. Isenburg, "Isotropic surface remeshing," in *Shape Modeling International, 2003*. IEEE, 2003, pp. 49–58.

[65] S. Fuhrmann, J. Ackermann, T. Kalbe, and M. Goesele, "Direct resampling for isotropic surface remeshing." in *VMV*. Citeseer, 2010, pp. 9–16.

[66] D. A. Field, "Laplacian smoothing and delaunay triangulations," *International Journal for Numerical Methods in Biomedical Engineering*, vol. 4, no. 6, pp. 709–712, 1988.

[67] T. Tachi, "Simulation of rigid origami," *Origami*, vol. 4, pp. 175–187, 2009.

[68] ——, "Rigid-foldable thick origami," 2011.

[69] B. J. Edmondson, R. J. Lang, S. P. Magleby, and L. L. Howell, "An offset panel technique for thick rigidily foldable origami," in *ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2014.

[70] S. A. Zirbel, R. J. Lang, M. W. Thomson, D. A. Sigel, P. E. Walkemeyer, B. P. Trease, S. P. Magleby, and L. L. Howell, "Accommodating thickness in origami-based deployable arrays," *Journal of Mechanical Design*, vol. 135, no. 11, p. 111005, 2013.

[71] M. R. Morgan, R. J. Lang, S. P. Magleby, and L. L. Howell, "Towards developing product applications of thick origami using the offset panel technique," *Mechanical Sciences*, vol. 7, no. 1, p. 69, 2016.

[72] Y. Zhou, S. Sueda, W. Matusik, and A. Shamir, "Boxelization: Folding 3d objects into boxes," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 71, 2014.

[73] K. Iwama and T. Nakashima, "An improved exact algorithm for cubic graph tsp," in *International Computing and Combinatorics Conference*. Springer, 2007, pp. 108–117.

[74] P. Vaněček and I. Kolingerová, "Comparison of triangle strips algorithms," *Computers & Graphics*, vol. 31, no. 1, pp. 100–118, 2007.

[75] G. Taubin, "Constructing hamiltonian triangle strips on quadrilateral meshes," in *Visualization and Mathematics III*. Springer, 2003, pp. 69–91.

[76] P. Diaz-Gutierrez and M. Gopi, "Quadrilateral and tetrahedral mesh stripification using 2-factor partitioning of the dual graph," *The Visual Computer*, vol. 21, no. 8-10, pp. 689–697, 2005.

[77] D. Cohen-Or and A. Kaufman, "Fundamentals of surface voxelization," *Graphical models and image processing*, vol. 57, no. 6, pp. 453–461, 1995.

[78] D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "Concorde tsp solver," 2006. [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde/index.html

Curriculum Vitae

Zhonghua Xi received his Bachelor of Engineering from Shanghai Jiao Tong University in 2009. He received his M.S. degree in Computer Science from George Mason University in 2015. His academic research is in the area of computational origami, computer graphics, motion planning and computer vision.

List of Peer-Reviewed Publications

1. Zhonghua Xi, and Jyh-Ming Lien. "Polyhedra Fabrication Through Mesh Convexification: A Study of Foldability of Nearly Convex Shapes," In: *International Design and Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE)* Columbus, OH, ASME, Aug. 2017.

2. Yanyan Lu, Zhonghua Xi, and Jyh-Ming Lien. "Online Collision Prediction Among 2D Polygonal and Articulated Obstacles," In: *International Journal of Robotics Research (IJRR)* 35.5 (Apr. 2016), pp. 476−500.

3. Guilin Liu, Zhonghua Xi, and Jyh-Ming Lien. "Nearly Convex Segmentation of Polyhedra Through Convex Ridge Separation," In: *Symposium on Solid & Physical Modeling (SPM); also appears in Journal of Computer-Aided Design.* Berlin, Germany, June 2016.

4. Huangxin Wang, Zhonghua Xi, Fei Li, and Songqing Chen. "Abusing Public Third-Party Services for EDoS Attacks," In: *10th USENIX Workshop on Offensive Technologies (WOOT).* Austin, Texas, Aug. 2016.

5. Zhonghua Xi, Yun-hyeong Kim, Young J. Kim, and Jyh-Ming Lien. "Learning to Segment and Unfold Polyhedral Mesh from Failures," In: *Shape Modeling International (SMI); also appears in Journal of Computers & Graphics.* Berlin, Germany, June 2016.

6. Zhonghua Xi and Jyh-Ming Lien. "Continuous Unfolding of Polyhedra - a Motion Planning Approach," In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Hamburg, Germany, Sept. 2015, pp. 32493254.

7. Zhonghua Xi and Jyh-Ming Lien. "Folding and Unfolding Origami Tessellation by Reusing Folding Path," In: *2015 IEEE International Conference on Robotics and Automation (ICRA).* Seattle, WA, May 2015.

8. Zhonghua Xi and Jyh-Ming Lien. "Plan Folding Motion for Rigid Origami via Discrete Domain Sampling," In: *2015 IEEE International Conference on Robotics and Automation (ICRA).* Seattle, WA, May 2015.

9. Guilin Liu, Zhonghua Xi, and Jyh-Ming Lien. "Dual-Space Decomposition of 2D Complex Shapes," In: *27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Columbus, OH: IEEE, June 2014.

10. Yanyan Lu, Zhonghua Xi, and Jyh-Ming Lien. "Collision Prediction Among Polygons with Arbitrary Shape and Unknown Motion," In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Chicago, IL, Sept. 2014.

11. Yanyan Lu, Zhonghua Xi, and Jyh-Ming Lien. "Collision Prediction: Conservative Advancement Among Obstacles With Unknown Motion," In: *International Design and Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE).* Buffalo, NY: ASME, Aug. 2014.

12. Yanyan Lu, Zhonghua Xi, and Jyh-Ming Lien. "Predict Collision Among Rigid and Articulated Obstacles with Unknown Motion," In: *The Eleventh International Workshop on the Algorithmic Foundations of Robotics (WAFR).* Istanbul, Turkey, Aug. 2014.

13. Zhonghua Xi and Jyh-Ming Lien. "Determine Distinct Shapes of Rigid Origami," In: *The 6th International Meeting on Origami in Science, Mathematics and Education (6OSME).* Tokyo, Japan, Aug. 2014.

14. Zhonghua Xi and Jyh-Ming Lien. "Folding Rigid Origami with Closure Constraints," In: *International Design and Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE).* Buffalo, NY: ASME, Aug. 2014.

15. Zhonghua Xi, Jyh-Ming Lien, Yi-Chang Chiu, and C. Y. David Yang. "Identify and Visualize Differences in Vehicle Trajectory Data," In: *7th International Visualization in Transportation Symposium.* Irvine, CA: TRB, Oct. 2013.