CURVED PATTERN ORIGAMI

A Thesis

by

HAN-WEI KUNG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Ergun Akleman
Committee Members,	Felice House
	Richard Furuta
Department Head,	Timothy McLaughlin

December 2014

Major Subject: Visualization

ABSTRACT

Origami is the art of paper folding. It transforms paper into a completed form through folding techniques. "ORI" means "folding" and "GAMI" means "paper" in Japanese. Japan has developed the most extensive tradition of origami, although the exact origin of origami remains unknown. In this thesis, I use a computer to design folding patterns, and then use a laser cutter to cut and etch the patterns in paper.

Firstly, I implemented a C++ application to define folding patterns and to save the patterns in EPS file format. I also used another C++ application developed by a PhD student Ozgur Gonen. This application permits us to create more complicated geometric patterns. Secondly, I exported the EPS file to an AutoCAD DWG file using Adobe Illustrator. Thirdly, I set up the AutoCAD file for the laser cutter, moving, resizing, and placing the drawing elements onto separate layers with different colors in AutoCAD. Finally, I operated the laser cutter to let laser cut through material.

The application that I implemented allows us to draw both straight lines and diagonals. However, we can arrange these lines only on a single square. In contrast, the application developed by Ozgur Gonen makes us able to arrange these lines on a diversity of geometries, such as triangles, rims, and polygons, though its diagonal-drawing function has not been implemented yet. Drawing folds using these applications has several advantages: scaling, stretching, and repetition are easy. Furthermore, we can always reload and modify any drawings.

To identify the performance requirements in our developed software solutions, I executed some of the crease patterns from the book "Folding Techniques for Designerss: From Sheet to Form" by Paul Jackson as our test cases. The test report will serve as a base for future studies on how to maintain and drive continuous quality

improvement in the software in meeting user requirements.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my committee chair, Dr. Ergun Akleman. He helped me come up with the thesis topic and diligently guided me throughout the course of writing this thesis. I thank him for responding to my questions patiently and promptly throughout my time as his student.

I also wish to thank my committee members, Dr. Richard Furuta and Prof. Felice House. I sincerely acknowledge their valuable comments on this thesis. During the difficult times when I defended this thesis, they gave me encouragement I needed to move on.

I would also like to thank Ozgur Gonen, who developed the graphics program *Shady*. Without his contribution of time and efforts, many of the origami models could not have been successfully produced.

Finally and foremostly I must thank my parents who raised me with unconditional love and supported me all these years. They have given up many things for me to study abroad. I also thank my sisters and brothers-in-law. Their unending support and faith in me during my good and bad times are so appreciated. This accomplishment would not have been achieved without them. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. MOTIVATION AND INTRODUCTION

In the sixth century A.D., Buddhist monks brought paper from China to Japan and Japan has elaborated origami to a very high art form. The Arabs the Moors were also able to make paper in the eighth century but it is unknown that they practiced paper folding or that they brought paper folding to Spain. It is known that origami is very popular in Spain, Germany, Italy, and South America.

Almost every flat material can be used for folding if it can hold a crease. We sometimes also need glue or tape to fix the material. Different kinds of materials affect the look of the finished sculpture. Normal copy paper or drawing paper is suitable for a wide range of models because they are light, flexible, and easy to cut and fold. Heavier weight paper (e.g., Bristol board) can be used for straight folds and large sculpture because it is thick and sturdy. Moreover, some kinds of paper are colored and have pretty patterns, adding a great appeal to the models. Fabric folding has fundamentally the same folding techniques but with fabric instead of paper. For example, napkin folding is one of the most popular kinds of fabric folding and is commonly seen in restaurants.

Origami models could resemble some shapes or forms in daily lives, such as boxes, hats, cups, and flowers. With a piece of paper, origami could reproduce them in an expressive way.

Practicing origami is fun, though it requires a lot of exercise. We would have to calculate the number of folds and the spacing between each fold. We should also be familiar with the texture of different kinds of materials. Only through constant experiments can we gain excellent skills and knowledge of origami. At the end, origami is not just for fun but also has many benefits like improving creativity, eye

hand coordination, geometry, measurement, and reasoning skills.

In this section I will motivate and introduce my research.

1.1 Motivation

Curved origami (see Fig. 1.1) is very different from conventional origami. In conventional origami, we fold along straight lines, while in curved origami, we fold along curves. The main difference is that the two opposite parts of the paper along the curved crease cannot lie flat against each other. Therefore paper bends itself into a natural equilibrium form.

Not many origamists have strived to create designs for curved-crease sculpture. One of the reasons is that it is hard to draw curved folds by hand. Another reason is that the results of curved models are usually hard to analyze and expect.

Since most of the existing software for origami design work with straight lines only, it is necessary to develop computational methods to design and construct complicated curved origami.

1.2 Introduction

I will now introduce the two programs we have developed and our procedure of making origami models. In this thesis, we have implemented two computer programs to design patterns for folding tessellations.

1. *OriPatch*: *OriPatch* is a C++ program for defining complicated folds on simple geometry. The user can use it to create straight lines, curves, and diagonals. With horizontal sliders, users can interactively modify some adjustable values (e.g., the number of folds) in the folding pattern. The limit of this program is that we are only allowed to create a single Bezier patch.

2. *Shady*: *Shady* is a C++ program for drawing simple folds on complicated

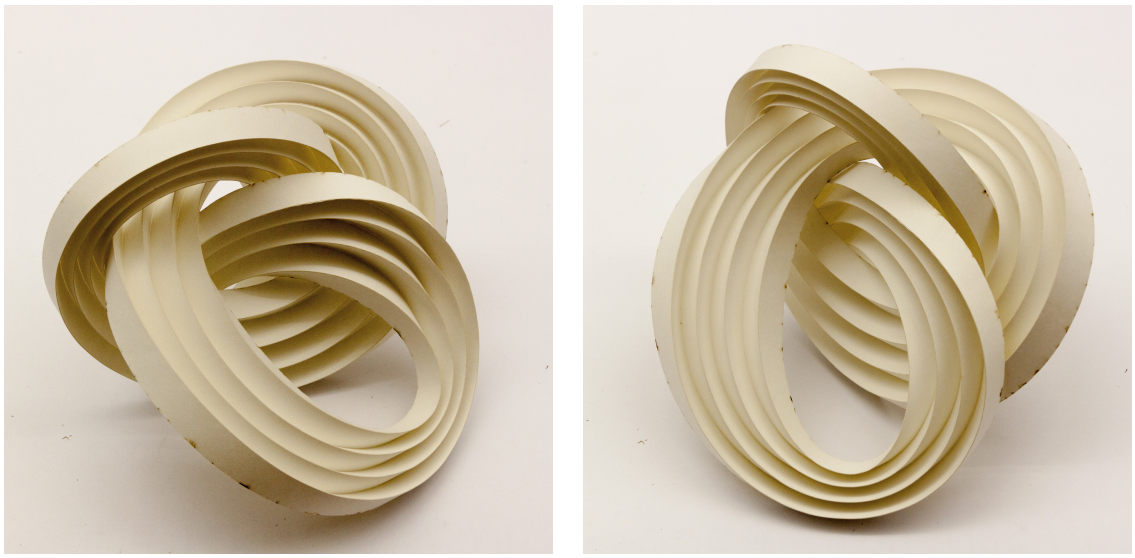


Figure 1.1: Curved origami sculptures

geometry. It supplies a number of tools, such as drawing straight lines, curves, multiple Bezier patches, except for diagonals. This program is more flexible to serve the needs of more advanced users.

We introduce a procedure of designing folds using computer software and producing origami models using a laser cutter. The first step in the procedure is to design and sketch out what we want to fold using our programs. Next, we export our drawing as a file type which the laser cutter can recognize. After that, we place the material we wish to engrave and cut onto the laser cutter bed and begin the laser engraving/cutting process. The final step involves hands-on activity. We fold along all the crease paths on both sides of the material and create an origami model.

To investigate if our programs satisfy the needs of origamists, I will further test our programs by executing some of the cases shown in Paul Jackson's book "Folding Techniques for Designers: From Sheet to Form. [14]" This book methodically

explains the key of folding techniques with clear instructions and inspirational diagrams. It begins with simple straight-line folds, including linear creases and pleat surfaces. As the designs become more complex, Jackson moves to more organic abstracts, such as curvature folds, crumpling and pinching. Therefore I will use our programs to accomplish some of the examples in this book, so that we can identify which examples can be implemented using our programs and which cannot.

For those patterns that cannot be produced using either *OriPatch* or *Shady*, possible solutions will also be discussed at the end. Some problems may be straightforward to be solved while others are fairly challenging, especially for the patterns that are composed of non-uniformly random geometries. A number of possible further studies could therefore concentrate on the enhancement of our programs. Using the same experimental set up and our programs, origami artists can be better equipped to tackle the challenge of designing for origami work.

2. RELATED WORK

I will describe some of the work related to origami. In 1989 at the first *International Meeting of Origami Science and Technology* Humiaki Huzita enumerated six different ways of defining a single fold by bringing together one or more combinations of points and lines on a sheet of paper [12]. Meanwhile Jacques Justin, in the same proceedings of that very first *Origami Science and Technology* conference, presented a paper in which he identified seven distinct operations of possible combinations [16]. In fact Justin's seven operations included Huzita's six operations and one extra operation which was not mentioned in Huzita's paper. However, Justin's investigation somehow remained unknown for many years. On the contrary Huzita's six operations became the foundation of the study of origami and was known as the Huzita Axioms. Afterwards in 2001 a Japanese origami specialist Koshiro Hatori found another way of creating a single crease that cannot be described in terms of any of Huzita's six operations [9]. Namely Hatori discovered the seventh axiom. Surprisingly it turned out that Huzita's six axioms plus Hatori's operation are actually Justin's seven operations in 1989. Later in 2006 Roger C. Alperin and Robert J. Lang proved that the seven axioms are complete combinations of the alighments [2]. The seven axioms have become known as the Huzita-Hatori axioms, which has been widely used in the study of origami constructions.

Several researchers have probed the mathematics of paper folding over the years. For instance, Marshall Bern and Barry Hayes showed that, if we are provided with a crease pattern and a mountain and valley assignment which guarantees the crease pattern can be folded to a flat origami, determining the correct layering ordering of a flat folded status is NP-hard [3]. More recently Demaine and Rourke presented a

thorough coverage of the mathematics of folding in Euclidean space. In their book "Geometric Folding Algorithms: Linkages, Origami, Polyhedra" [6] they divided folding problems into three categories: linkages (one-dimensional objects), paper (two-dimensional objects), and polyhedra (three-dimensional objects). Then they emphasized the algorithms and computational aspects for each of these categories.

There have been some approaches for synthesizing crease patterns that can produce three-dimensional origami models. For example, the leading origami artist and theorist Robert J. Lang has investigated methods for designing flat-folded uniaxial bases [21]. The base can be shaped by adjusting the lengths of the flaps. He developed the tree algorithm [19, 20]. It allows users to design an origami base that topologically resembles a tree graph with preferred edge lengths, which can be folded into an origami sculpture. Lang's methods are suitable for designing organic models, such as insects, animals, and other living forms. Fig. 2.1 shows some of his origami works.

(a) Bull Moose, 2002 (b) Giraffe, 2011 (c) Beachcomber, 2013

Figure 2.1: Examples of Robert J. Lang's origami works.

Tomohiro Tachi proposed a method for simulating rigid origami and developed a system for simulating folding process from an origami crease pattern to a final base fold, using rigid origami simulation [28]. Fig. 2.2 shows the interface of this system. He studied the rigid deployable structures and proposed computational

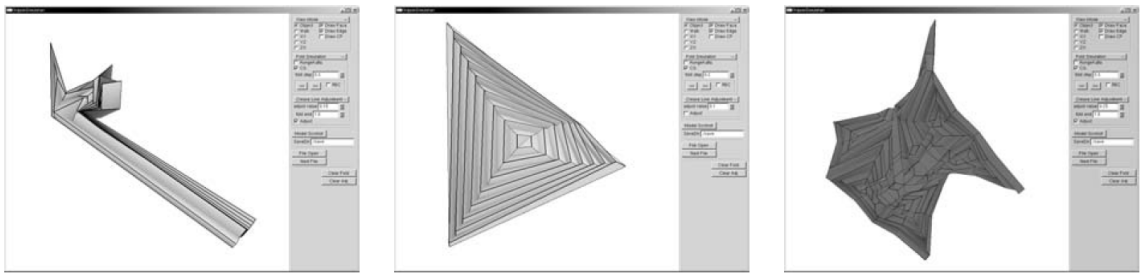


Figure 2.2: Screenshots of the rigid origami simulation program developed by Tachi [28].

methods [27, 29] to acquire variations of Miura-ori-like quadrilateral mesh origami that preserves rigid-foldability and flat-foldability. He further developed the software program *Origamizer* [5] which can efficiently generate a crease pattern for an arbitrary polygonal model. In *Origamizer* origami structure is represented by describing geometric properties, such as the positions of vertices and the lengths of creases. A user can determine vertices and creases at any arbitrary position and orientation, and therefore manage to generate crease patterns for very complex three-dimensional forms

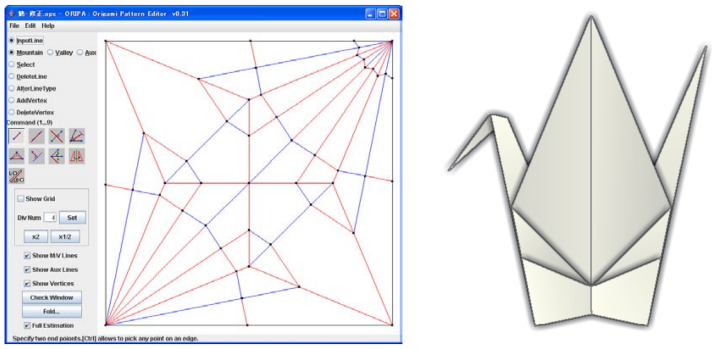
Jun Mitani et al. [7, 24] proposed computational rendering techniques to construct an origami model using a matrix that represents the overlapping relation between two faces. At that time he [23] developed an application, *ORIPA*, which renders a folded paper shape from a crease pattern inputted by the user. Fig. 2.3 displays the application window of *ORIPA* and its rendered origami result.

Other tools have also been created to support the origami design process. For example, the software *Doodle* [13] translates the source code written in its special descriptive language into an origami diagram that illustrates how folds should be made. In addition, Akitaya et al. [1] proposed an algorithm which generates a corresponding diagram (step-by-step folding sequences) for a crease pattern, accelerating

the process of drawing step-by-step instructions. *Snap-n-Fold* developed by Zhu et al. [30] gives users natural interface to generate folding patterns. Users can use mobile devices to take pictures of real-life objects and then use *Snap-n-Fold* to create folding patterns of the objects. Fig. 2.4 shows two example results of using the *Snap-n-Fold* tool.

Several researchers have further employed origami-inspired structure to engineering applications. One of the early studies is Takeo Kanade's "Origami World Theory" and skew symmetry [17]. Kanade introduced the "Origami World" for handling region-segmented images. It is a surface-oriented model where using the skew symmetry heuristic to determine the orientation of planes in space becomes more straightforward than other known methods. Kanade's approach has contributed to the field of computer vision over years. Lately Greenberg et al. [8] presented four natural phenomena of paper folding, with their corresponding dynamics and graph models, to demonstrate the analogy between origami models and spherical lamina emergent mechanisms. Moreover Lee et al. [22] applied the magic-ball crease pattern to design a wheel robot that can be printed on paper and then be folded and shaped into its final shape.

Other systems integrate our physical world with the power of computational models. For instance, *Easigami* [10, 11] is a tangible user interface and embedded interactive system. It allows users to model polyhedral objects by connecting and folding flat polygonal pieces. Refer to Fig. 2.5 to see how *Easigami* can be used to create a truncated tetrahedron. Furthermore, Chang et al. [4] presented a 3D *Origami Simulator* with multi-touch interaction. Users can interact with virtual 3D origami objects with multi-touch and physics simulation. Fig. 2.6 displays the *Origami Simulator* with a multi-touch interface. Finally, *Origami Desk* [15] provides users with multiple modes of interfacing with itself in creating an origami box or an



(a) (b)

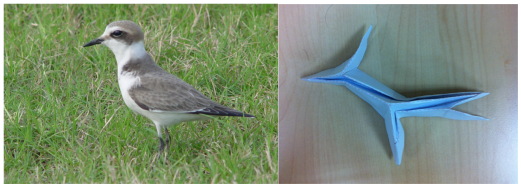
Figure 2.3: (a) *ORIPA* interface with the crease pattern of a crane (b) Shaded representation of origami work in *ORIPA*. This application is developed by Mitani [23].

origami crane. The desk projects video clips to instruct users how to fold origami paper. Animations are also projected onto users' paper to show how to make folds. The system also utilizes electric field sensing to determine the location of the touch inputs on the desk surface, and swept frequency sensing to extract a fold.

A few studies have discussed the computation of curved folding. Some interactive systems for designing curved folds have also been proposed. For example, Kilian et al [18] presented a computational framework for representing curved folding using planar quad mesh. Mitani [25] proposed a method for designing curved origami on two principal types of rotational geometry: cylinders and cones. Mitani and Igarashi [26] also proposed an interactive system to design planar curved folds using mirror reflection through an implicitly defined plane.

While there have been several systems built for making folding patterns, most of them focus on traditional origami, where creases are straight lines. As a consequence, there is a need for innovative tools that allow users to design curved folding patterns. In this research, we have implemented two applications which focus on facilitating the process of designing curved origami patterns.

(a) User selects horse in the image **(b)** Origami Base generated based on the selected horse



(a) User selects bird in the image **(b)** Origami Base generated based on the selected bird

Figure 2.4: Users take photos of real-life objects and use *Snap-n-Fold* [30] to generate origami models from the photos.

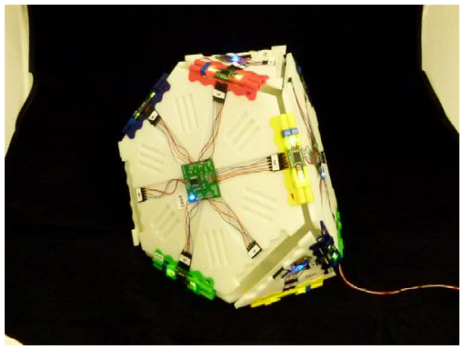


Figure 2.5: The user completes a polyhedron using the *Easigami* TUI [10, 11].

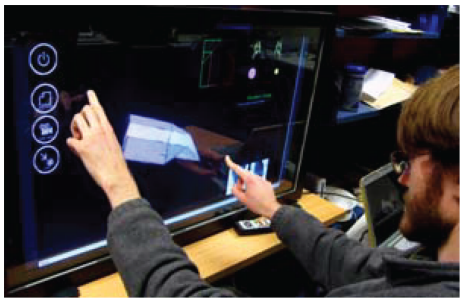


Figure 2.6: A participant interacts with the *Origami Simulator* [4].

3. METHODOLOGY

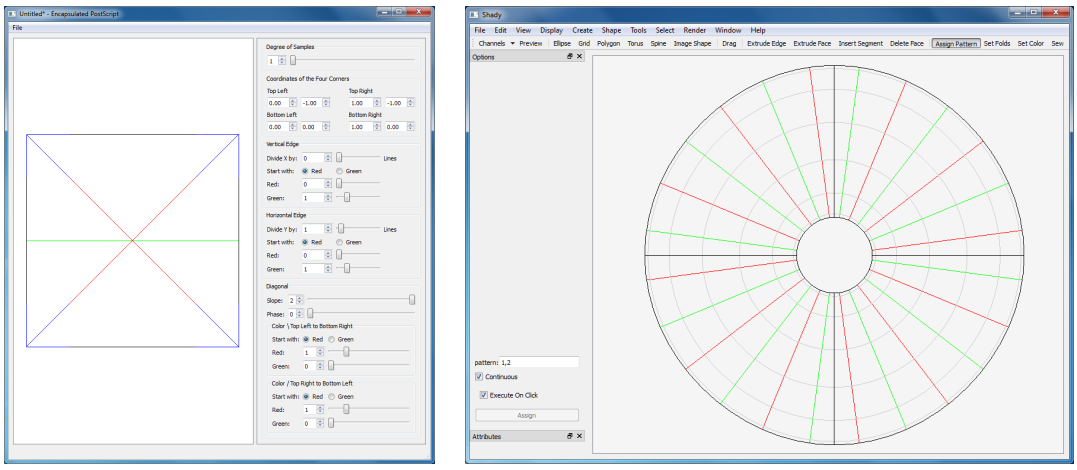
I will now introduce the research methodology for this study. To employ modern technology in producing our origami, we follow a procedure that consists of the following four steps:

1. Design Curved Origami Shapes: Currently, there is few software to design curved origami patterns. Therefore, we developed two 2D drawing programs to generate curved crease patterns for designing curved origami. The interfaces and representational powers of these two programs are different as shown in Figure 3.1.

   The first program, *OriPatch*, allows users to design curved origami using one single cubic tensor-product Bezier patch. Therefore, *OriPatch* provides a grid structure topologically. Each of the curved folds in *OriPatch* is obtained using either u, v or $u+v$ constant curve. The simplicity of underlying structure helps us to define folding patterns simply using sliders.

   The second program *Shady* is developed to provide more general shapes and structures for origami. With *Shady*, it is possible to create more complicated structures as planar 2-manifold surfaces, in which every face is a cubic tensor-product Bezier patch. To create triangles we simply move two control points to the same position. This allows us to construct a complicated structure that visually consists of only quadrilaterals and triangles with curved boundaries. In *Shady*, we still use u or v constant curves to obtain folds. However, unlike *OriPatch* in this case u or v patterns cannot necessarily be consistent globally. Therefore, we cannot provide same type of global interface to obtain patterns. Instead, we let users locally assign patterns using local u and v constant curves.

Because of this limitation, it is also hard to assign diagonal patterns that correspond to $u + v$ constant curves.



(a) *OriPatch* Interface (b) *Shady* Interface

Figure 3.1: Examples from interfaces of two software we have developed. (a) shows an example of origami design from *OriPatch* interface. In *OriPatch* users define curved folding structures using a single cubic Bezier patch. The u, v coordinates of the Bezier patch is used to create folding curves. Users adjust the attributes of the folding pattern curves using a set of horizontal sliders. (b) shows an example of origami design from *Shady* interface. *Shady* allows users to design origami patterns using multiple cubic Bezier patches. One of the fundamental features of *Shady* is that it allows users to create a variety of planar 2-manifold structures that consist of multiple Bezier patches. Examples of such structures include toroidal shapes, polar structures, and grids. *Shady* also provides a number of operations over the 2-manifold structures. In *Shady*, folding patterns are assigned by using a number pattern instead of sliders.

2. Prepare Files for Laser Cutting: The second step in the procedure is to set up a file for laser cutting so that the laser cutter can engrave and cut through a material according to the file information. In our case, we use *OriPatch* and *Shady* to export our artwork in EPS file format, and then use third-party software to convert and format the file for laser cutting. The laser cutter determines whether to cut or etch an edge based on which layer that contains

the edge. At this point, we have a proper file format with correct layers which are assigned to proper edges. We also scale the drawing pattern appropriately using the third-party software in order to make it fit the laser cutter bed.

3. Laser Etch/Cut Materials: Once we have our file ready for laser cutting, the next step is to bring laser-safe materials to the laser cutter to engrave and cut the artwork. Almost all paper products (normal copy paper, cardboard, mat board, Bristol board, etc.) are generally safe materials.

4. Fold the Model: The final stage is folding the necessary creases and then collapsing the material along the fold lines into the final model.

Using the above procedure, we are able to design and create a wide variety of origami models.

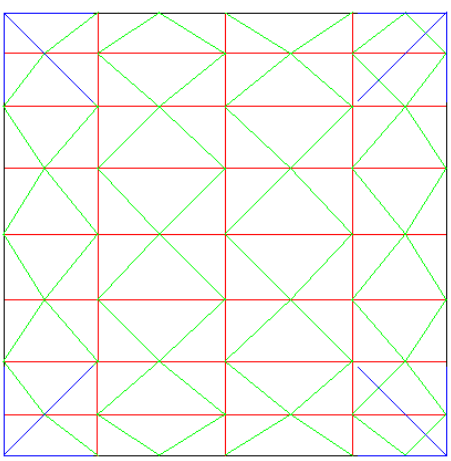
4. CURVED ORIGAMI DESIGN: ORIPATCH

The following section details the features of *OriPatch*. The program *OriPatch* enables us to define curved crease patterns using one single cubic tensor-product Bezier surface patch. A cubic tensor-product Bezier surface patch can be constructed by a grid of sixteen control points. Then the curved creases are given by the u, v coordinates of these control points on the Bezier surface patch. In *OriPatch* we are able to edit the color and the number of curved creases by adjusting the horizontal sliders. We can thus divide a square into strips in horizontal, vertical, and diagonal directions, to create complex forms (see Fig. 4.1) We are also able to change the way each line segment bends by dragging the blue control handle on each corner of the square.

The common technique to model a smooth surface is to apply a Bezier surface. Since a cubic Bezier surface creates a smooth and continuous surface, I utilized a cubic Bezier surface as the basis of the drawing model in *OriPatch*. A cubic Bezier surface represents the region bounded by a set of closed Bezier curves. To explain cubic Bezier surfaces, I will begin with cubic Bezier curves because they are easier to understand and implement. Next I will discuss Bezier surfaces, whcih are generalizations of Bezier curves to higher dimensions. At the end of the chapter we cover the basics of the *OriPatch* user interface.

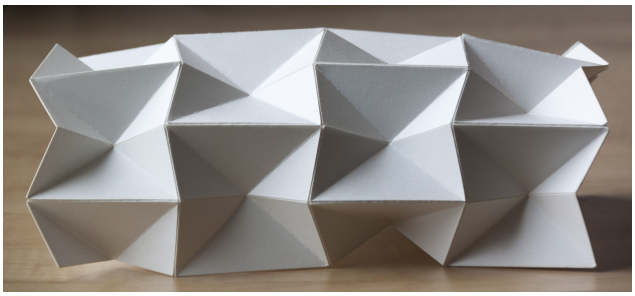
4.1 Bezier Curve

In our case, we used a cubic Bezier curve that is defined by four control points. Two of the four control points are endpoints or anchor points. They determine the starting point (x_0, y_0) and destination point (x_3, y_3) of the curve. The rest of the control points, (x_1, y_1) and (x_2, y_2), are handles, which influence the shape of the

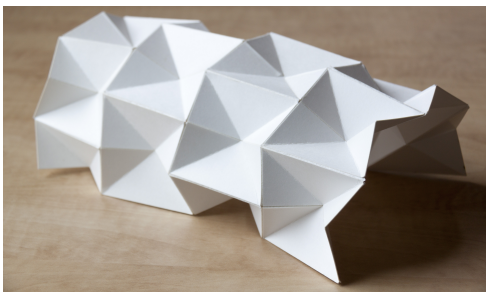


(a) Folding Pattern Design

(b) Top View



(c) Front View



(d) Three-quarter View

Figure 4.1: An example of an origami model created by using the program *OriPatch*. (a) Design of folding patterns in *OriPatch*. The green-color line segments are valley-fold and the red-color line segments are mountain-fold. (b), (c), and (d) are the top, front, and three-quarter view of the finished origami model.

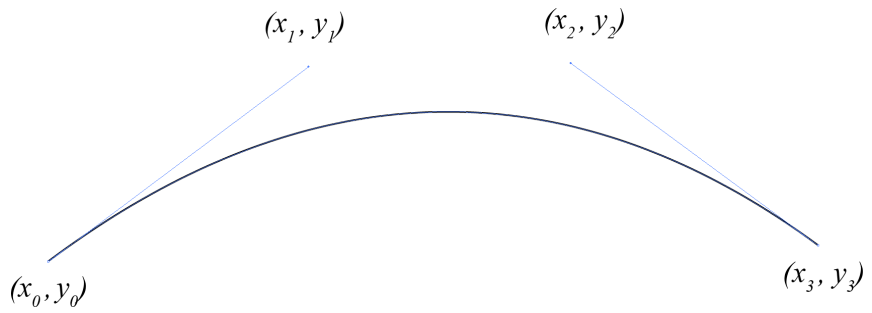


Figure 4.2: For a given value of u, a point on a cubic Bezier curve can be defined as the weightd sum of these four control points, (x_0, y_0), (x_1, y_1), (x_2, y_2), and (x_3, y_3). That is, $BezierCurve(u) = (1-u)^3(x_0, y_0) + 3(1-u)^2u(x_1, y_1) + 3(1-u)u^2(x_2, y_2) + u^3(x_3, y_3)$

curve. Fig. 4.2 illustrates a cubic Bezier curve that is determined by four control points (x_0, y_0), (x_1, y_1), (x_2, y_2), and (x_3, y_3).

Equation 4.1 and Equation 4.2 are the parametric equations for the Bezier curve $BezierCurve(u) = \{x(u), y(u)\}$.

$$x(u) = \sum_{i=0}^{3} B_i^3(u)x_i \tag{4.1}$$

$$y(u) = \sum_{i=0}^{3} B_i^3(u)y_i \tag{4.2}$$

where B_i^m is a Bernstein polynomial, using Equation 4.3 below.

$$B_i^m(u) = \binom{m}{i} u^i(1-u)^{m-i} \tag{4.3}$$

Therefore, to calculate the coordinates of a point on the curve at a specific parameter value u $(0 \leq u \leq 1)$, we solve for x and y by substituting Equation 4.3 into

Equation 4.1 and Equation 4.2, as seen below.

$$
\begin{aligned}
x(u) &= \sum_{i=0}^{3} B_i^3(u) x_i \\
&= B_0^3(u)x_0 + B_1^3(u)x_1 + B_2^3(u)x_2 + B_3^3(u)x_3 \\
&= \binom{3}{0} u^0 (1-u)^3 x_0 + \binom{3}{1} u^1 (1-u)^2 x_1 + \binom{3}{2} u^2 (1-u)^1 x_2 + \binom{3}{3} u^3 (1-u)^0 x_3 \\
&= (1-u)^3 x_0 + 3(1-u)^2 u x_1 + 3(1-u)u^2 x_2 + u^3 x_3
\end{aligned}
$$

$$
\begin{aligned}
y(u) &= \sum_{i=0}^{3} B_i^3(u) y_i \\
&= B_0^3(u)y_0 + B_1^3(u)y_1 + B_2^3(u)y_2 + B_3^3(u)y_3 \\
&= \binom{3}{0} u^0 (1-u)^3 y_0 + \binom{3}{1} u^1 (1-u)^2 y_1 + \binom{3}{2} u^2 (1-u)^1 y_2 + \binom{3}{3} u^3 (1-u)^0 y_3 \\
&= (1-u)^3 y_0 + 3(1-u)^2 u y_1 + 3(1-u)u^2 y_2 + u^3 y_3
\end{aligned}
$$

where

$$0 \leq u \leq 1$$

4.2 Bezier Surface

A Bezier surface is formed as the Cartesian product of two orthogonal Bezier curves.

In *OriPatch* I applied a tensor-product Bezier surface patch of degree $(3, 3)$. Fig. 4.3 illustrates a single bicubic Bezier surface patch that is defined by a set of 16

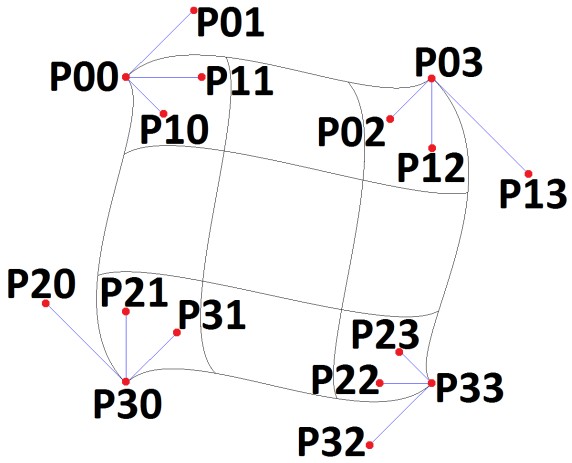


Figure 4.3: A tensor-product Bezier surface patch of degree $(3, 3)$. The u and v coordinates of the control points p_{00} through p_{33} determine the curvature and locations of creases and cusps of the surface patch. In other words, given 16 control points, we can generate a cubic Bezier surface patch using $BezierSurface(u, v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(u)B_j^3(v)p_{ij}$.

control points. The parametric equation of the surface is given as

$$BezierSurface(u, v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(u)B_j^3(v)p_{ij} = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(u)B_j^3(v)(x_{ij}, y_{ij})$$

Recall that B_i^m is a Bernstein polynomial, as seen in Equation 4.3. For each of the equations $x(u, v)$ and $y(u, v)$ defined in the interval $u, v \in [0, 1]$, a point on the Bezier surface is given by

$$x(u, v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(u)B_j^3(v)x_{ij}$$

$$y(u, v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(u)B_j^3(v)y_{ij}$$

where x_{ij} and y_{ij} are the x and y values at row i and column j of the 4×4 Bezier geometry matrix, as seen in Table 4.1.

$$p_{ij} = (x_{ij}, y_{ij}) = \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \\ p_{30} & p_{31} & p_{32} & p_{33} \end{bmatrix}$$

Table 4.1: A 4×4 Bezier Geometry Matrix

4.3 *OriPatch* Interface Overview

Fig. 4.4 shows the complete user interface of *OriPatch*. The toolbox on the right side of the interface consists of four group boxes: the *Degree of Samples* box, the *Coordinates of Four Corners* box, the *Vertical Edge and Horizontal Edge* box, and the *Diagonal* box. Refer to Fig. 4.5 to see each of these boxes.

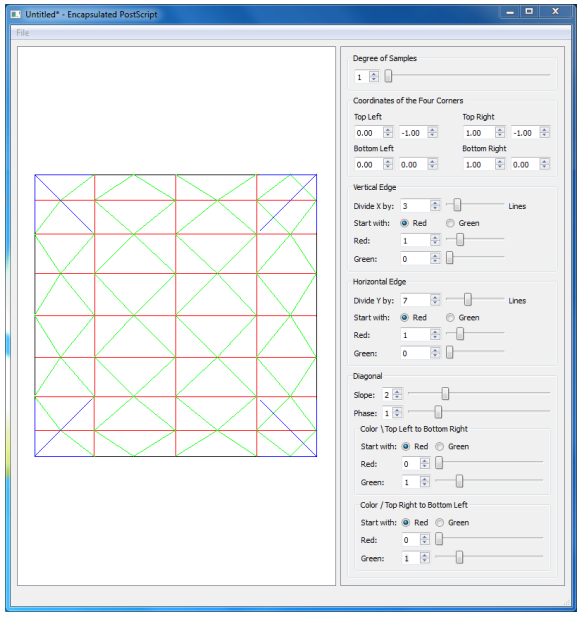
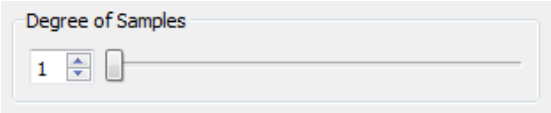


Figure 4.4: *OriPatch* interface

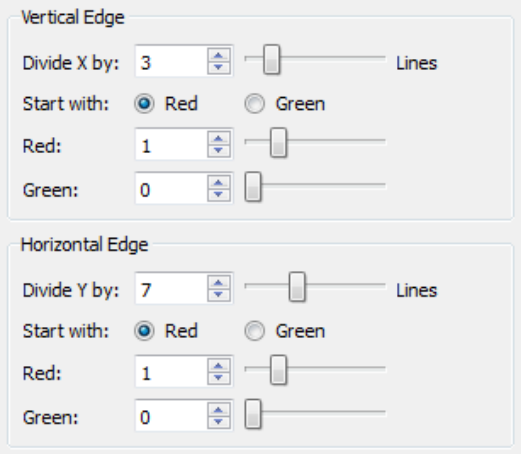
The following features are available to *OriPatch* users.

1. The *Degree of Samples* box contains one slider which determines how smooth the creases are. The slider controls a variable of an integer type which serves as the number of samples along each line segment. We interpolate the u and

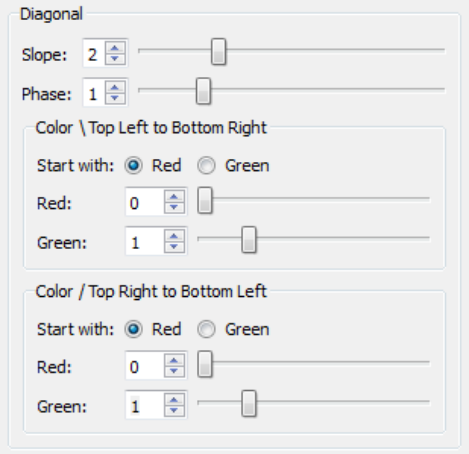


(a) *Degree of Samples* box

(b) *Coordinates of Four Corners* box



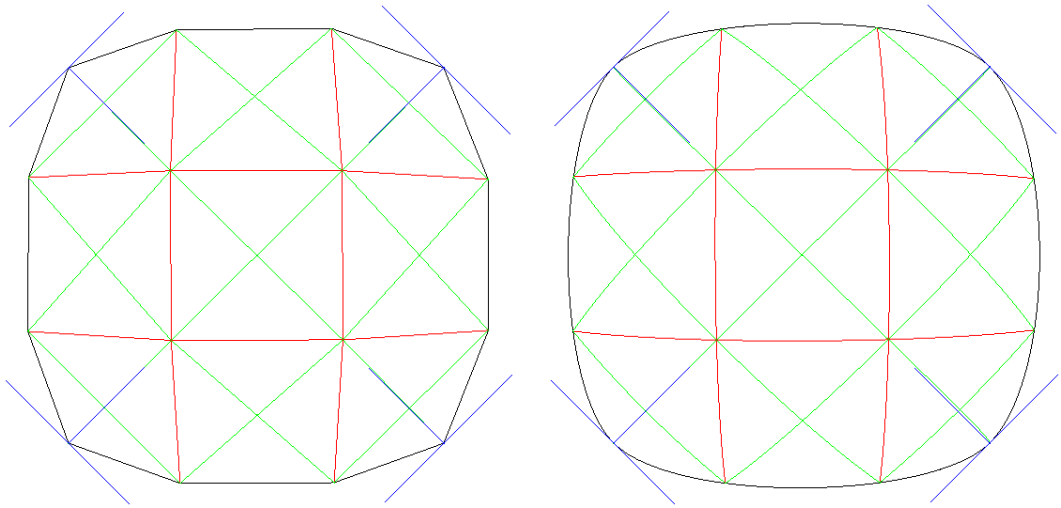
(c) *Vertical Edge and Horizontal Edge* box



(d) *Diagonal* box

Figure 4.5: Demonstrations of designing the folding patterns as shown in Fig. 4.4. (a) The *Degree of Samples* box shows that we have only one sample point for every line segment because we work with straight lines in this case. (b) The *Coordinates of Four Corners* box displays the four coordinates of each of the quad's corners. We can enter numeric values with the keyboard for these coordinates. (c) The *Vertical Edge and Horizontal Edge* box shows that the quad is divided into fourths along the x-axis and is divided into eighths along the y-axis, using mountain-fold (red) creases, across the quad. (d) The *Diagonal* box shows that the slope of all diagonals is 2 because for every 1 grid unit that the diagonal travels in x-direction, 2 grid units are traveled in y-direction. Next, the phase shift is 1 because every diagonal at the right column is shifted one grid unit upward. All diagonals use valley-fold (green) creases across the quad.

v coordinates provided by the control points so that we calculate each of the position of the sample points. Then *OriPatch* draws a smooth line through



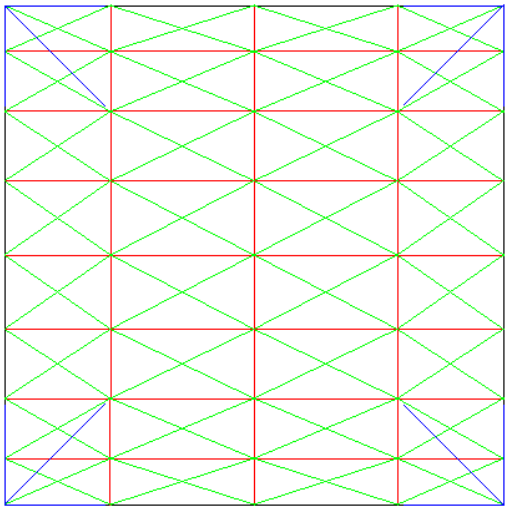
(a) (b)

Figure 4.6: The more the samples are, the smoother the line segments become: (a) A cubic Bezier surface patch with 1 sample along each line segment, (b) A cubic Bezier surface patch with 20 samples along each line segment

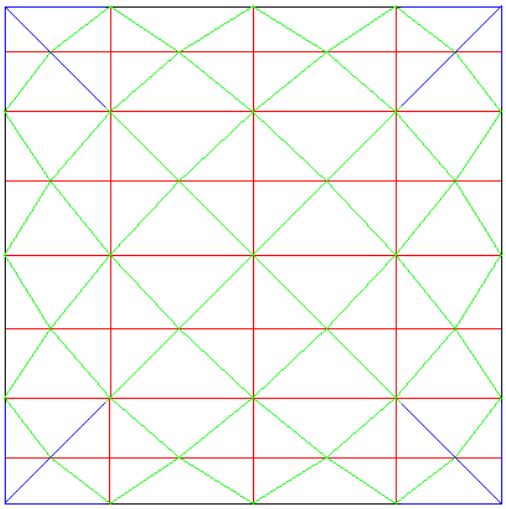
these sample points. Fig. 4.6 illustrates the effects of increasing the number of samples per line segment generates a smoother edge.

2. The *Coordinates of Four Corners* box provides four pairs of floating point spin boxes which specify the coordinates of the cusps of the Bezier surface patch. The eight numbers represent x and y coordinates of the upper-left, upper-right, lower-left and lower-right corners, respectively. These spin boxes give users full controls to edit the coordinates of the four corners rather than just graphically dragging those corners. In addition to clicking and dragging the cusps to the location where we want our new cusps to be, we can precisely type in the x and y coordinates of the cusps.

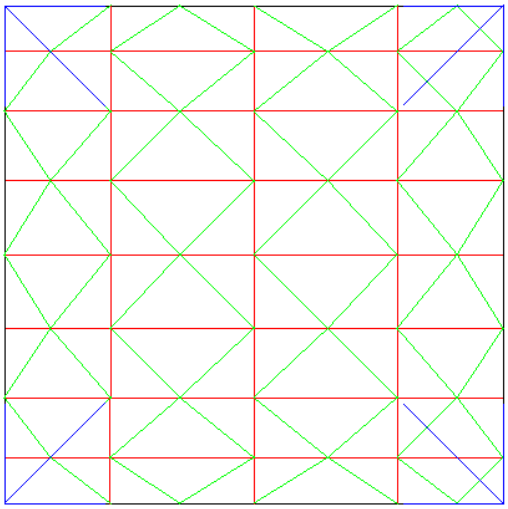
3. The *Vertical Edge and Horizontal Edge* box contains three sliders for each of the vertical and horizontal directions respectively. For vertical edges, one of the three sliders controls the number of divisions in vertical direction while



(a) A cubic Bezier surface patch with diagonals' slope 1 and phase shift 0.



(b) A cubic Bezier surface patch with diagonals' slope 2 and phase shift 0.



(c) A cubic Bezier surface patch with diagonals' slope 2 and phase shift 1.

Figure 4.7: Illustration of cubic Bezier surface patches with different slopes and phase shifts of diagonals.

two of them determine the color of each vertical line segment. Identically, for horizontal edges, one of the three sliders controls the number of divisions in horizontal direction while two of them determine the color of each horizontal line segment.

4. The *Diagonal* box provides six sliders. One of the sliders specifies the phase shit of diagonals (see Fig. 4.7.) The phase shift is the difference, expressed in the number of grid units, between two columns having the same motif. Another slider controls the slope of diagonals. The slope is the change in the vertical difference of a diagonal on the grid structure over the change in horizontal difference. The rest two pairs of sliders let us work with the colors of diagonals going top-left to bottom-right direction and the colors of diagonals going top-right to bottom-left direction.

5. CURVED ORIGAMI DESIGN: SHADY

In this following section I will describe the features of our second program *Shady*. *Shady* is a program for designing a variety of complex geometric folding patterns. It enables us to draw not only squares but also polygons (e.g., pentagons and hexagons), circles, and any freeform shapes. Furthermore, we can create a triangle by first creating a four-vertex object, then moving two of the four vertices to the same position to form a triangle. In *Shady* the connected planar shapes in two-manifolds are expressed as a union of cubic tensor product Bezier patches. One of the advantages of using quad patches to form these shapes is that we can represent complicated geometric structures that are composed of only quadrilaterals and triangles bounded by a set of Bezier curves.

We still calculate the u and v coordinates for interpolating a Bezier curve as we did in *OriPatch*. However, in *Shady* we cannot guarantee that u and v coordinates are always compatible with each other globally. Hence we have difficulties designing the same global interface to define folding patterns as we did for *OriPatch*. To solve this problem, in Shady we let users assign patterns to each two-manifold surface individually by using local u and v coordinates. It is also difficult to make the divisions parallel to diagonals due to this restraint. Fig. 5.1 demonstrates an incompatible uv coordinate systems.

In this section we present the three main toolsets in our program *Shady*. Fig. 5.2 displays an overview of the user interface.

1. *Ellipse*, *Grid*, *Polygon*, *Torus*, *Spine*, and *Image Shape* tools: Using these tools, we can design different kinds of geometry including circles, triangles, squares, pentagons, and hexagons etc. For example, we can create an easy square using

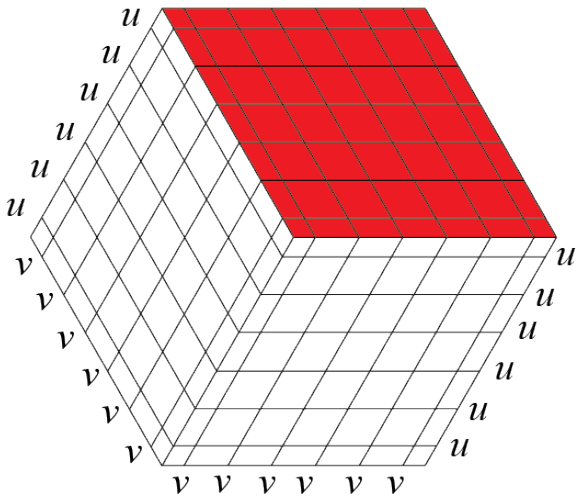
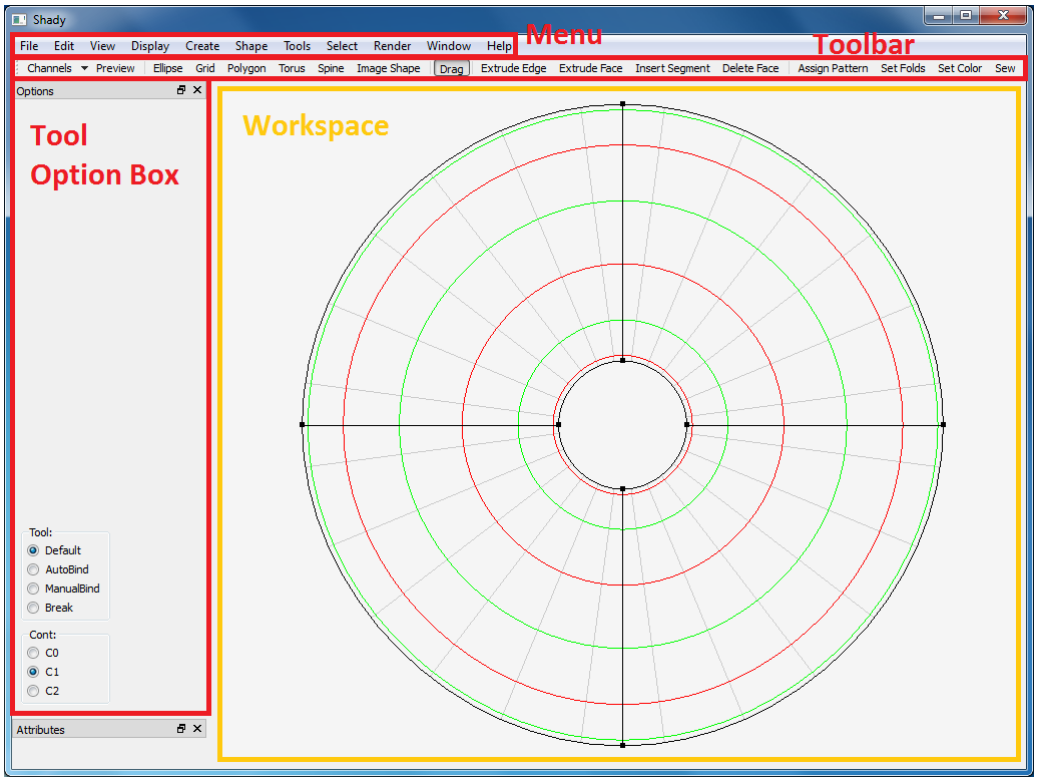


Figure 5.1: An example of how u and v coordinates lead to contradictions on the red patch. This contradiction prevents us from assigning u and v coordinates globally.

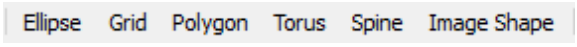
a simple procedure done with the Grid Tool and a few adjustments in the option box. Similarly, to create a square donut, we can use the Torus Tool and set the attributes in the option box to become square.

2. *Drag*, *Extrude Edge*, *Extrude Face*, *Insert Segment*, and *Delete Face* tools: This toolset lets us to edit the geometry. The Drag Tool enables us to move the handles to deform the geometry. Further, we can add more geometry to an existing geometry using the Extrude feature. We can extrude polygon edges or faces using the feature. For example, when we extrude the exterior edges, it creates new connecting faces outwards along the edges. In addition, the Insert Segment Tool lets us split one or more polygon faces in a geometry by clicking an exterior edge of a face to specify the location of the split. Last, we can also delete a face in order to create a hole in the geometry using the Delete Face feature.

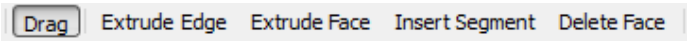
3. *Assign Pattern* and *Set Folds* tools: The Assign Pattern feature lets us quickly



(a) *Shady* interface.



(b) *Ellipse*, *Grid*, *Polygon*, *Torus*, *Spine*, and *Image Shape* tools



(c) *Drag*, *Extrude Edge*, *Extrude Face*, *Insert Segment*, and *Delete Face* tools

(d) *Assign Pattern* and *Set Folds* tools

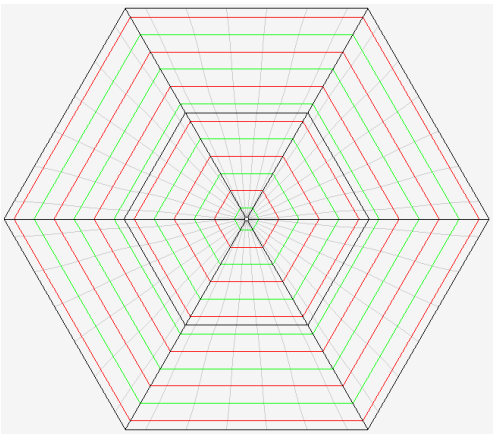
Figure 5.2: A brief summary of the main *Shady* interface (a) The following are the components of the *Shady* interface: *Menu*, *Toolbar*, *Tool Option Box*, and *Worksapce*. The toolbar contains three fundamental toolsets, as seen in (b), (c), and (d).

assign colors to folds so that we are able to apply the red marks to make mountain folds and the green marks to make valley folds all at once. We can associate each fold with a different color so that it is easier to identify which

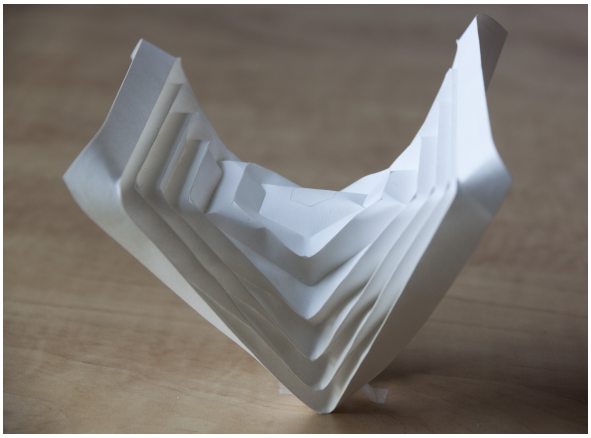
folds are mountain folds or valley folds. Moreover, we can use the Set Folds feature to divide a polygon face into two or more equal parts. This is useful when we need to divide paper into two or more lengths or angles.

Fig. 5.3 demonstrates how to design a hexagonal parabola using *Shady*. To create a hexagon, first on the toolbar, click *Torus*, and the torus tool option box appears in the side bar. Then type 6 to set the *Sides* value in the option box. Next turn off the *Keep Tangents Smooth* option to generate a hexagon with sharp corners. Finally click *Insert* to create a hexagon. To make creases which alternate mountain-valley-mountaion-valley... across the hexagon, first on the toolbar, click *Assign Pattern*. Then manually type in the *pattern* value of "$1, 2$" in the option box and press *Enter*. Finally click each of the six edges of the hexagon to mark mountain-valley-mountaion-valley... creases.

Fig. 5.4 demonstrates another example of drawing a shape with a freeform outline using *Shady*. To create a freeform shape, first click *Spine* on the toolbar. Then click on the workspace to create a path with straight segments. To close the path, hover over another point and right-click to close the path. Next, simply click *Insert* in the spine tool option box in the side bar to produce the freeform shape. Moreover, the *Drag* tool on the toolbar lets us change the shape of this freeform object by dragging any of the cusp points. Similarly, to make creases which alternate mountain-valley-mountaion-valley... across the freeform shape, we use the same workflow as we apply for creating a hexagon, as seen in Fig. 5.3. First go to *Assign Pattern* on the toolbar. Then enter numerical values of "$1, 2$" in the option box in the side bar and press *Enter*. Finally click each of the edges that outline the freeform shape to mark mountain-valley-mountaion-valley... creases.

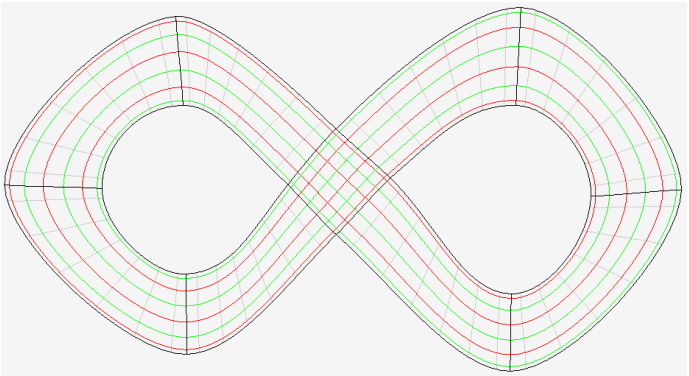


(a) Use *Shady* to design a hexagon shape.

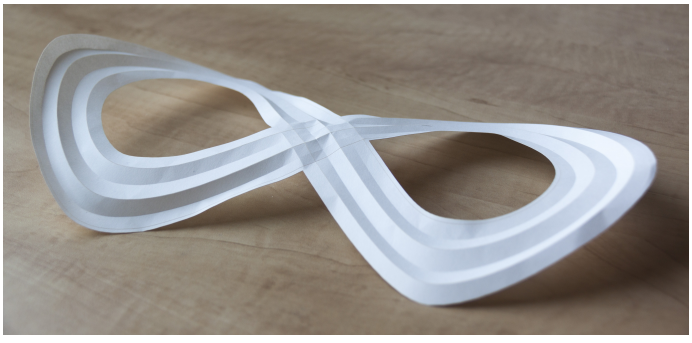


(b) A hexagonal parabola made from the hexagon shape.

Figure 5.3: A hexagonal parabola created by using the program *Shady*.



(a) Use *Shady* to design a shape with a freeform outline.



(b) A freeform origami model made from the freeform shape.

Figure 5.4: A freeform origami model created by using the program *Shady*.

6. PROCESS AND IMPLEMENTATION

This section summarizes our process of making curved origami models.

6.1 Design Curved Origami Shapes

First, we use our programs, *OriPatch* and *Shady*, to design folding patterns. We are able to make crease patterns that include not only straight lines but curves. As the user designs an initial origami geometry, a simple patch, such as a strip or a circle, is usable. Then the user can deform the geometry and specify the number and color of the folds on the geometry. In our software, red lines represent mountain folds and green lines represent valley folds. As we finish the design, we save it in an EPS format using our software.

6.2 Prepare Files for Laser Cutting

To laser cut our files, we can use files from different kinds of software as long as we can export or save as a vector-format file type. In our case, we use AutoCAD, which saves vector files as DWG. The vector format is the only format the laser cutter understands. A vector file is a graphics file that contains vector graphics only, rather than raster graphics. Vector graphics uses geometrical primitives and mathematical formulas to represent images. Therefore these images that are made up by points, lines, curves, and shapes are more flexible because they are fully resizable and stretchable. On the contrary, raster graphics represents images through pixels. Hence raster images are less flexible because they always appear burry when scaled up.

Before we create a lasercutting file using AutoCAD, we first convert the EPS file from our programs to a DWG file using Adobe Illustrator. Fig. 6.1 shows the artwork

in *Shady*, as seen in Fig. 3.1 (b), brought from EPS files into Adobe Illustrator.

Here is a typical workflow to export an EPS file as a DWG file in Adobe Illustrator.

1. Click on *File* in the menubar when Adobe Illustrator loads, then click *Export* in the drop-down menu.

2. Select a location for the file, enter a filename, select the *AutoCAD Drawing (DWG)* format, and then click *Save*.

3. Set the AutoCAD export options. In our case, we first choose AutoCAD Version 2004/2005/2006. We enter values of 1's for *Scale Units* and disable *Scale Lineweights* option. Next we have 8 colors for the color depth of the exported file and choose PNG for *Raster File Format*. Last we select *Maximum Editability* and disable the rest of other options. The DXF/DWG Options dialog box is shown in Fig.6.2.

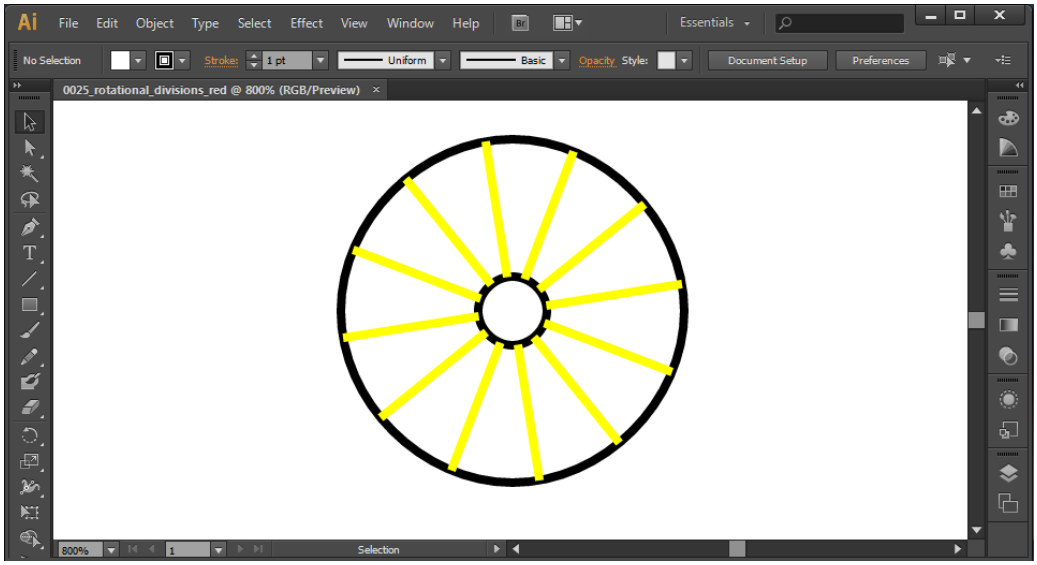


Figure 6.1: An example of an EPS file opened in Adobe illustrator.

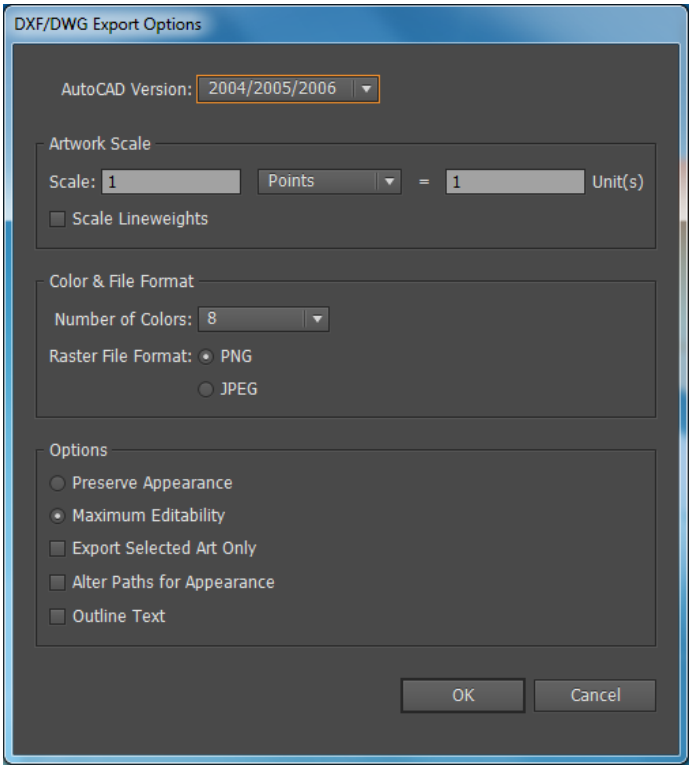


Figure 6.2: Adobe Illustrator AutoCAD Export Options dialog box gives us control over AutoCAD-format (.dwg or .dxf) images which may be supplied when we export a document to AutoCAD formats.

Once we have created our DWG file, we are ready to format it in AutoCAD and then send the file to the laser cutter. Fig. 6.3 displays a DWG file opened in AutoCAD.

The first thing we should do is to decide what material we want to cut our artwork from. Once we know our material size, we can make our artwork fit within that area. We may set our file up to the sheet size our material comes in, though it is not necessary to use the full sheet of material. Recall that when printing with paper, we include a small margin in the file. Likewise, as we layout our design, we should keep in mind that we leave a small gap between the design and the edge of the material. In our case, we use a laser cutter with a $32'' \times 18''$ bed and a margin of ${}^{1}/_{4}''$ around all margins. Therefore if the given sheet size is $32'' \times 18''$, then the largest safe design

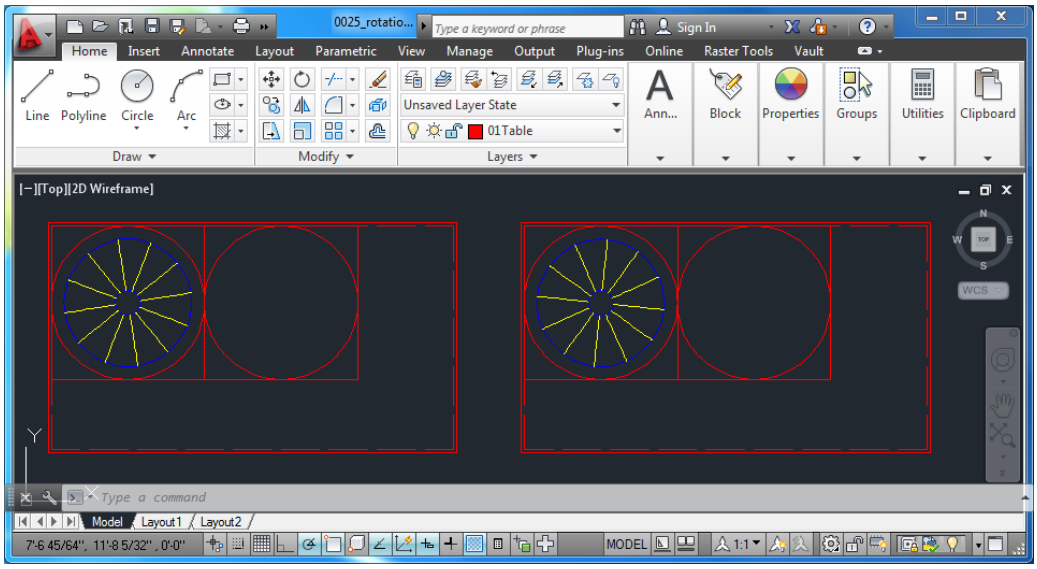


Figure 6.3: An example of exporting the artwork from Adobe Illustrator, as seen in Fig. 6.1, and then importing it to AutoCAD.

we can cut on this sheet size is $31.5'' \times 17.5''$.

The second step will be scaling our drawing to make sure that it can fit the sheet size. This following is intended to serve as a guide on how to scale our design size to the size of the sheet you are cutting in AutoCAD.

1. Type "scale" into the command prompt and press *Enter*.

2. Select the objects that need to be scaled and press *Enter*.

3. Specify a base point.

4. Type "r" and press *Enter* to use the "Reference" option.

5. Choose two points on the object that need to be scaled to the desired length.

6. Type "p" and press *Enter* to use the "Points" option.

7. Choose two points that establish the desired length.

Next, we set up layers with RGB top-bar colors and move each line the laser cutter associates with each action to each layer with the corresponding color. In our case, the laser cutter only accepts 2 colors from our files. Blue is used for all lines to be cut through the material, and yellow is used for all lines associated with scoring or engraving. The laser cutter will ignore any other colors supplied in the file.

6.3 Laser Etch/Cut Materials

Now that we know our design fits safely within our material and all lines are associated with correct actions, we choose appropriate materials (i.e., paper or board) and use the laser cutter to cut the material.

7. RESULTS AND CONCLUSIONS

7.1 Making Origami Models

For this thesis, *Shady* was used to generate most of the folds, and *OriPatch* was used to define diagonal folds. Then Adobe Illustrator was used to convert our folding patterns (EPS) into AutoCAD formats (DWG.) Laser cutting file preparation was done in AutoCAD, with some layout setup to accommodate the desired origami model size, and eventually we cut the origami model out of a laser cutter.

I used folds of parallel straight lines, parallel curves, radial straight lines, and radial curves as my initial test cases. The first case is dividing paper using a set of parallel straight lines while the second case is dividing paper using a set of parallel curves. The length between the mountain-valley pairings may differ from the one between the valley-mountain pairings, although in our cases the lengths between every line and every curve are equally spaced. In the same way, the third case is dividing paper using a set of radial straight lines while the fourth case is dividing paper using a set of parallel curves. Likewise the angle between the mountain-valley pairings may differ from the one between the valley-mountain pairings.

Fig. 7.1 displays each of these four crease patterns and its corresponding result. Fig. 7.1.(a) is a crease pattern with a collection of equally spaced parallel straight lines. Red lines represent mountain folds while green lines represent valley folds. Using such pattern, we can fold a strip of paper along these parallel straight lines and then glue one end to the other end to create a simple cylinder, as seen in Fig. 7.1.(b). Additionally Fig. 7.1.(c) is a crease pattern with a collection of equally spaced parallel curves. Fig. 7.1.(d) is a strip of paper that is folded along these curves and then be glued one end to the other to make a cylinder. Fig. 7.1.(e) is a

crease pattern with a set of radial straight lines that are arranged around a rim. All lines rotate by equal angles. We can apply this pattern to a rim of paper, folding along these radial straight lines to shape a cone, as seen in Fig. 7.1.(f). Similarly Fig. 7.1.(g) is a crease pattern with a set of radial curves that are arranged around a rim. All curves rotate by equal angles. Fig. 7.1.(h) is a rim of paper that is folded along these curves.

These four cases are the foundations on which more complex work can be built. Once the four basics have been learned, it is possible to create a lot of variations. Fig. 7.2 shows some more complex origami work. Fig. 7.2.(a) is a rim with consecutive mountain and valley folds that are all concentric circles. When the creases are folded, the sheet becomes structurally complex and can be twisted into many different shapes, as seen in Fig. 7.2.(b). Fig. 7.2.(c) is a quadrilateral that is divided by parallel equally spaced S-shaped curves. Further a zig-zag line is arranged across the quadrilateral. When a piece of paper is folded based on this pattern, it bent itself into a roof-like form, as seen in Fig. 7.2.(d). Similarly in many respects, Fig. 7.2.(e) is also a quadrilateral divided by parallel equally spaced S-shaped curves, except that there are two zig-zag lines that divide the quadrilateral into thirds. With this pattern, we can bend and flex a sheet to make a tunnel-like object, as seen in Fig. 7.2(f).

7.2 Software Testing

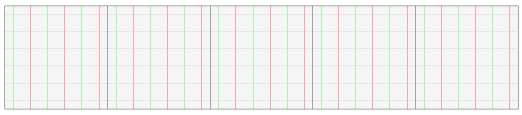
To evaluate the capability of our programs and determine if they can provide patterns that are used by origami artists, I used drawings in the book "Folding Techniques for Designers: From Sheet to Form" by Paul Jackson. The results of the software test can be seen in Fig. 7.4 to Fig. 7.9. It turned out that *Shady* is more applicable than *OriPatch* in terms of accomplishing such a wide variety of examples in this book.

There are some patterns from the book that cannot be obtained using either *Ori-Patch* or *Shady*. However, it is possible to extend some capabilities of the programs to obtain these patterns. For example, we cannot use *Shady* to create the pattern as shown in Fig. 7.3.(a). However we can create a pattern, as seen in Fig. 7.3.(b), which is similar to the original pattern in many respects, except that the folds in one of the two triangles do not rotate as expected. Thus we can obtain the original pattern if we have the option to control the orientation of folds in the triangle. Take Fig. 7.3.(b) for example. If we can change the orientation of the folds in the upper-left triangle by 120 degrees, we can make the pattern almost the same as the original. Afterward, we can acquire the pattern as seen in Fig. 7.3.(c) by simply decreasing the number of divisions in each triangle.

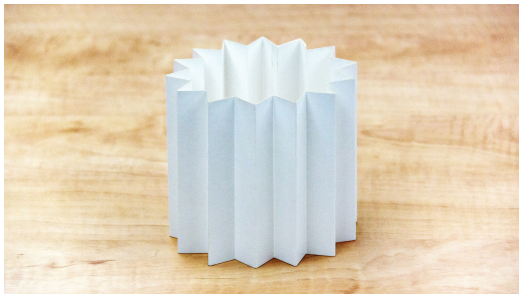
Apart from this, we can create a broad variety of patterns if we can move and sew edges back together. The move and sew feature is useful for quickly joining together separate geometric shapes. For instance, it can combine separate triangles along their selected border edges by moving one selected triangle and merging the selected edges together so that one quadrilateral results. Following this further, we can join together triangles and quadrilaterals to produce crease patterns such as the ones in Fig. 7.9 by selecting the edges we want to join and then executing the feature.

7.3 Conclusions

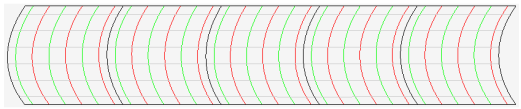
Complex crease patterns could recommend future work for us to extend the attributes of our program. In future work we will follow up on the experience gained from this initial study to further mature our program. Our long-term goal is to eventually bring a program for origamists to design crease patterns procedurally. Consequently they can use a computer to draw folding patterns easily rather than use geometry equipment to draw them on paper by hand.



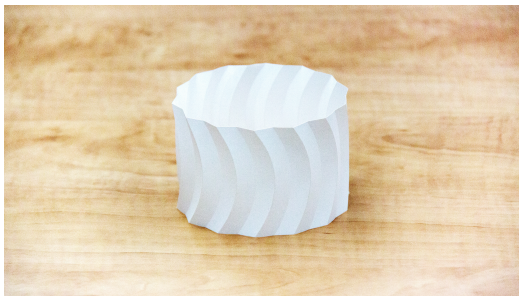
(a)



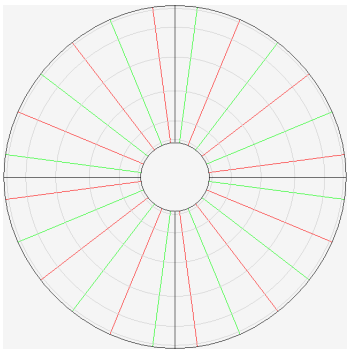
(b)



(c)

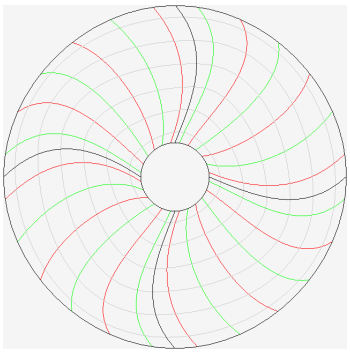


(d)



(e)

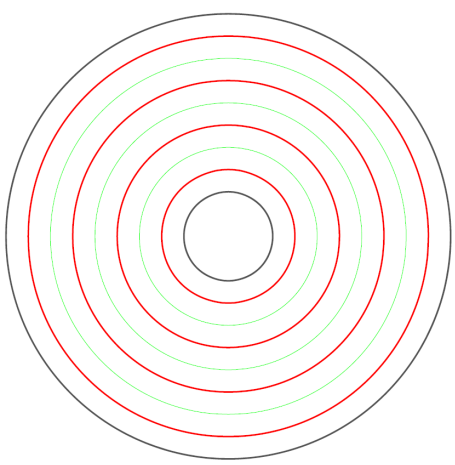
(f)



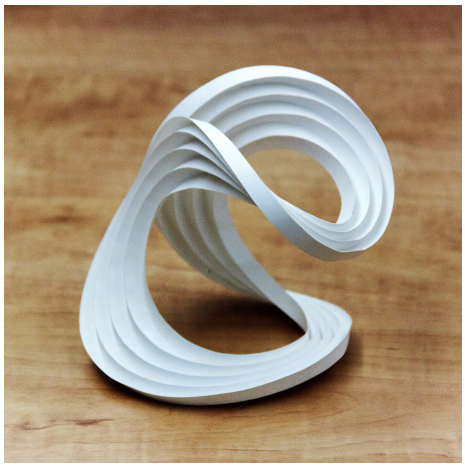
(g)

(h)

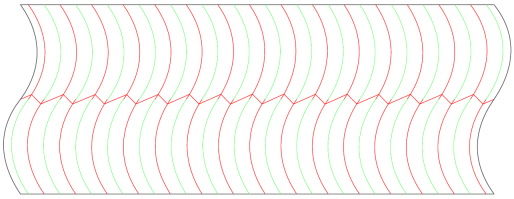
Figure 7.1: Four basic folded designs



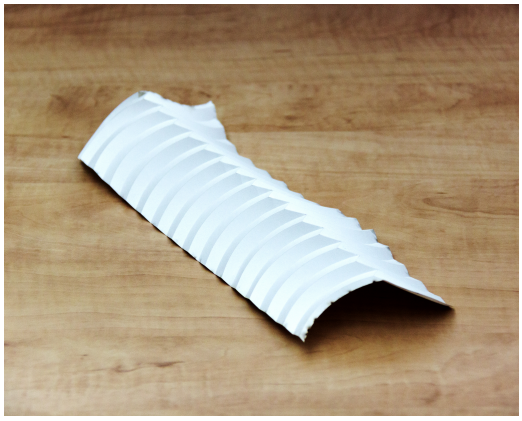
(a)



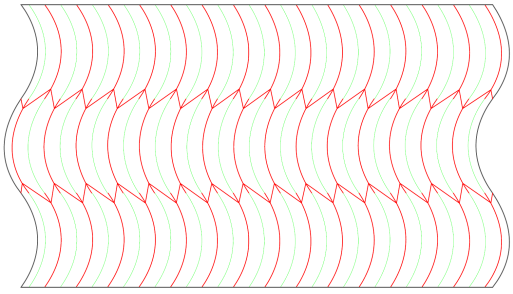
(b)



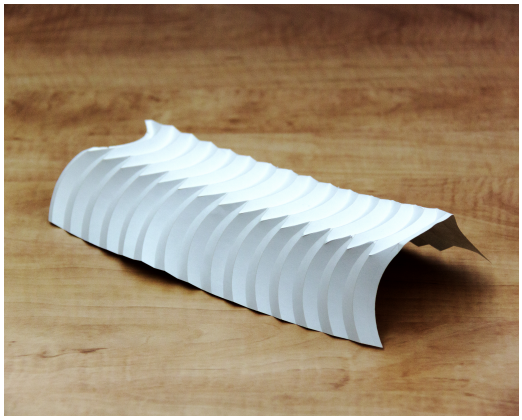
(c)



(d)

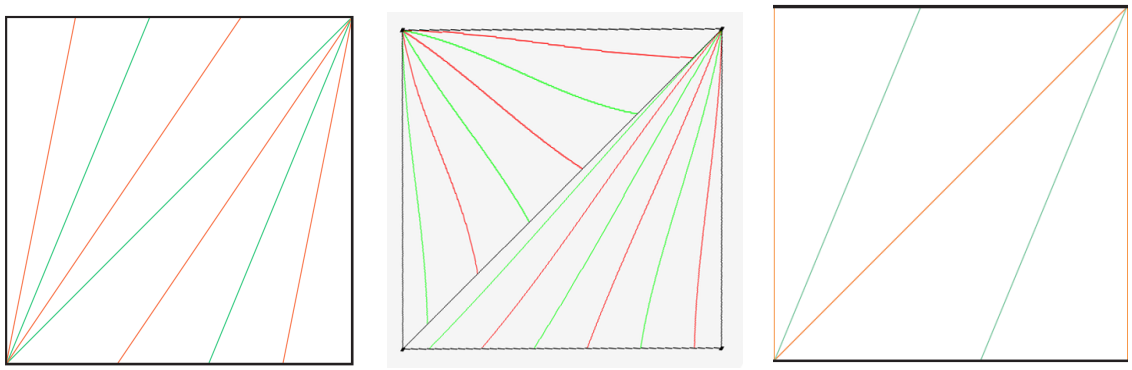


(e)



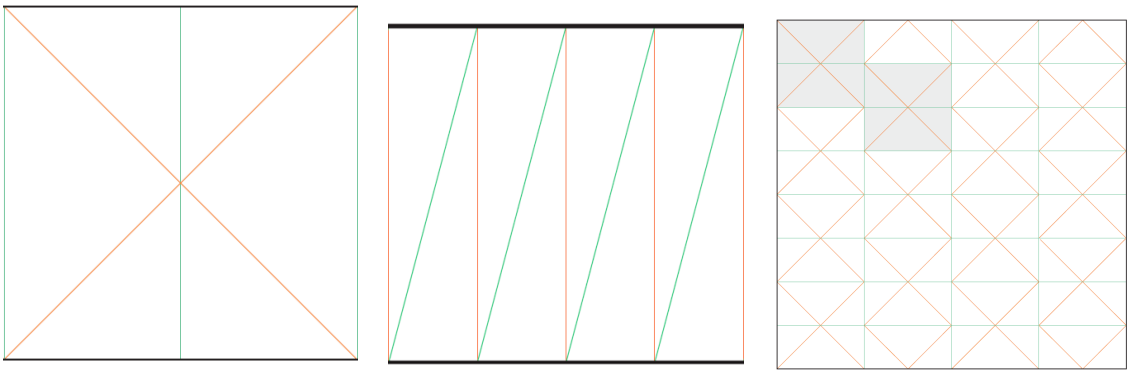
(f)

Figure 7.2: Complex origami



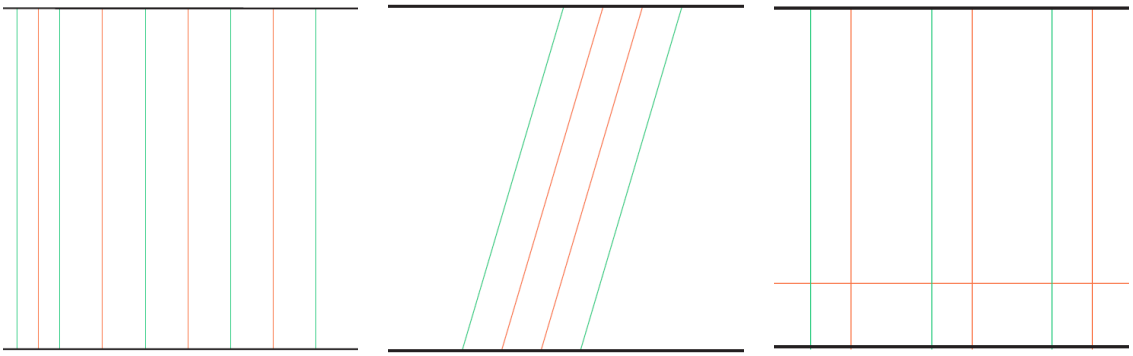
(a) (b) (c)

Figure 7.3: Although we have not been able to use *Shady* to directly generate the pattern that is exactly the same as the one in (a), we can create pattern (b) which resembles (a). To further make (b) look more similar to (a), we can change the orientation of the folds in the upper-left triangle by 120 degrees. Furthermore (c) can be acquired by decreasing the number of divisions in both upper-left and lower-right triangles.



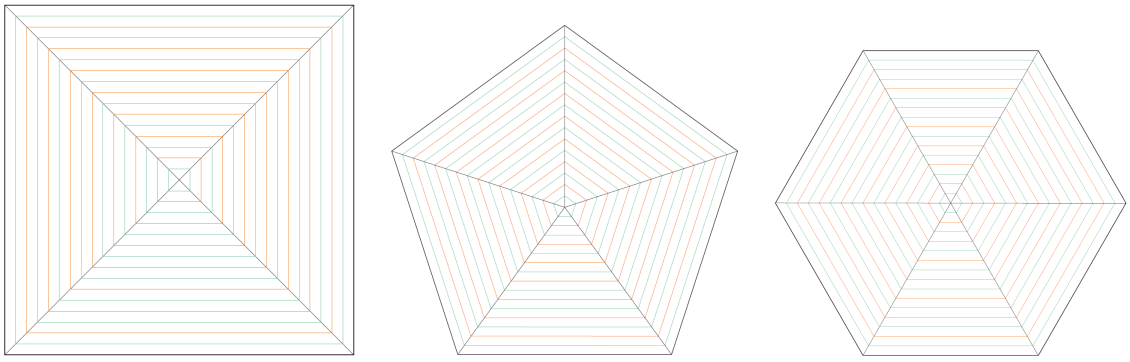
(a) (b) (c)

Figure 7.4: Examples of the crease patterns that can be made by using *OriPatch*. Users can also use *Shady* to make patterns (a) and (b). However, to be able to generate pattern (c) using *Shady*, we should additionally provide a tool for connecting two existing vertices. Take a square for example. A user can use this tool to draw an edge between two corners so the square has two triangles in it.



(a) (b) (c)

Figure 7.5: Examples of the crease patterns that can be made by using *Shady* only. Users cannot make these patterns using *OriPatch*. One of the reasons why users are unable to define these patterns in *OriPatch* is that explicit control of the lengths between edges are not provided. In *OriPatch*, a quadrilateral is divided into equally spaced lengths always. Such manner prevents users from altering either of the two endpoints of an edge and creating the pattern as shown in (b). To be able to define these patterns, *OriPatch* must provide a function for the user to move a vertex along an edge without shifting the edge itself.



(a) (b) (c)

Figure 7.6: Examples of the crease patterns that can be made by using *Shady* only. Users cannot make these patterns using OriPatch. OriPatch provides a user with only one single Bezier quad to design a folding pattern. Therefore users cannot create other kinds of geometric shapes, such as pentagons and hexagons. To aid a user in the attempt to easily form a composite geometry object, we should have a tool which performs stitching together multiple objects. We should also provide a feature that allows a user to scale, translate, rotate, and duplicate targeted objects to create an array of them.

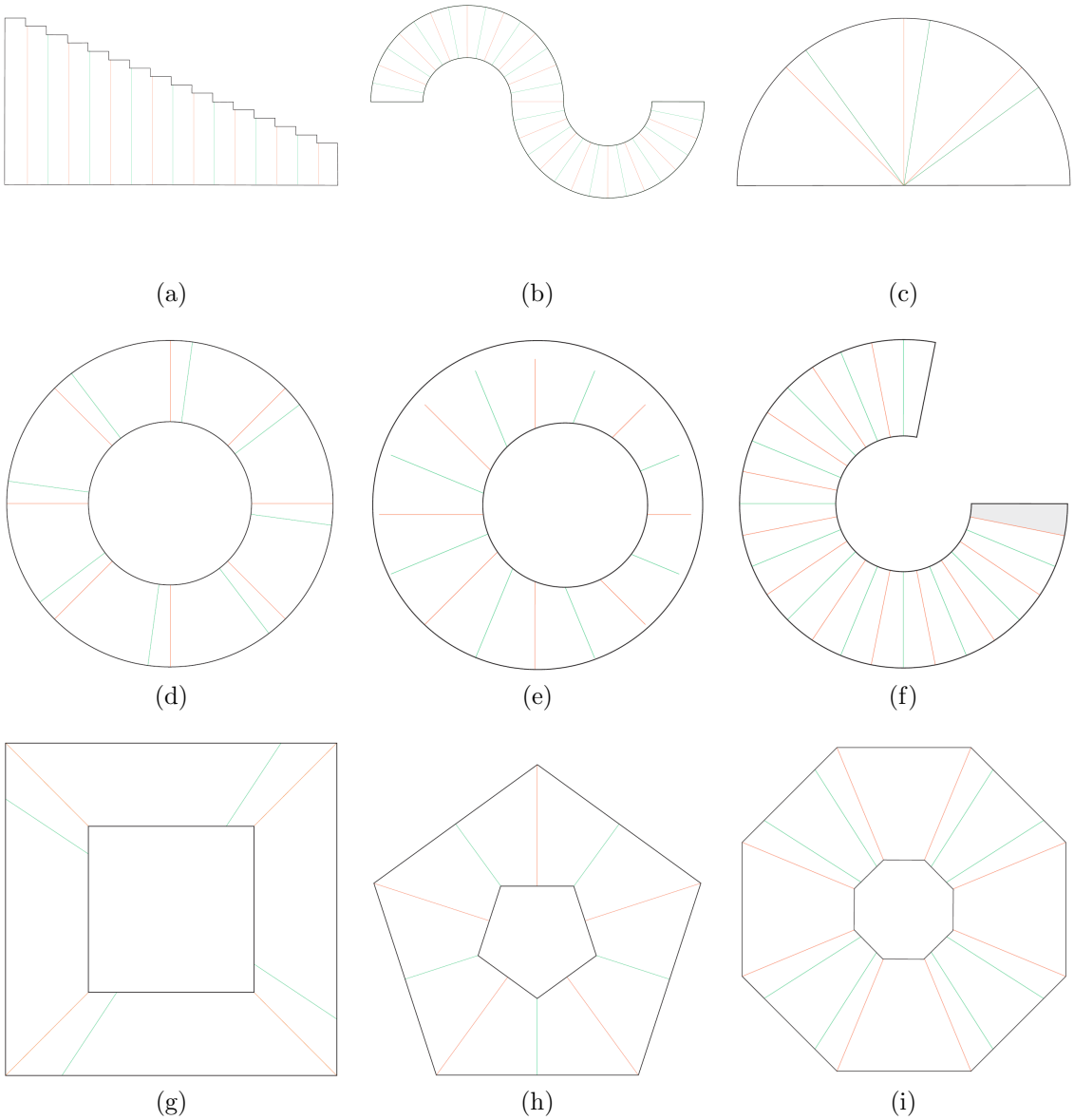


Figure 7.7: Examples of the crease patterns that can be made by using *Shady* only. Users cannot make these patterns using *OriPatch*. *OriPatch* allows users to define a crease pattern using only one single cubic tensor-product Bezier quad. Rims, circles and any types of polygons with more than 4 sides cannot be generated in OriPatch. To let users create different kinds of geometric shapes, we should provide a user interface for the user to handle multiple cubic Bezier quads. Further a merge boundry edge tool option should be provided so that the user can use it to merge or weld faces together.

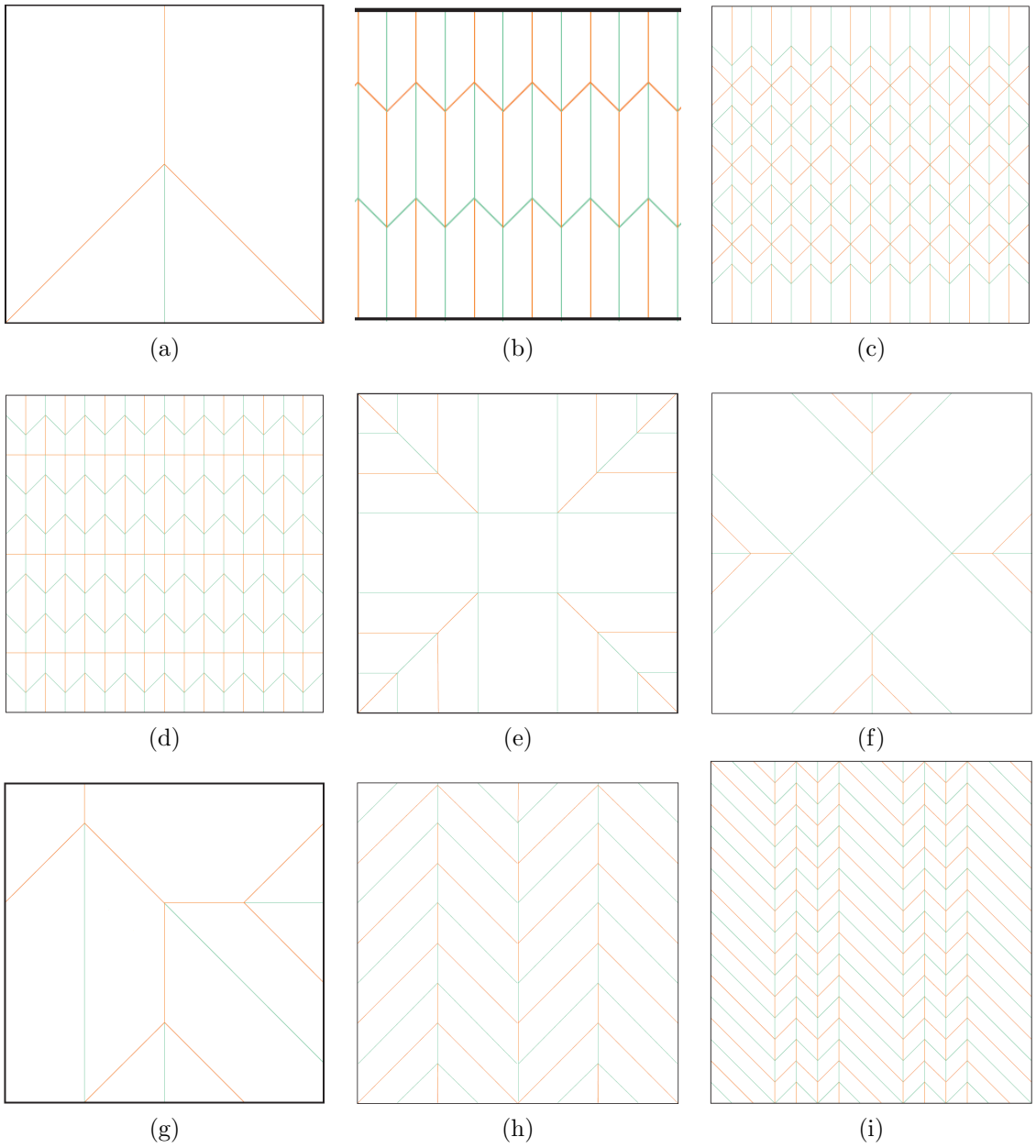


Figure 7.8: Examples of the crease patterns that can be made by using *Shady* only. Users cannot make these patterns using *OriPatch*. When users define folds on a quad using *OriPatch*, all folds are arranged across the entire quad vertically, horizontally, and/or diagonally. Users are forbidden from breaking and deleting these folds. To let users create these patterns, we must provide an insert edge loop tool for users to select and then split a face across an edge ring on the quad. Users can also use this tool to divide an edge into a series of smaller edges. Moreover we should support a feature which allows users to remove unwanted edges.

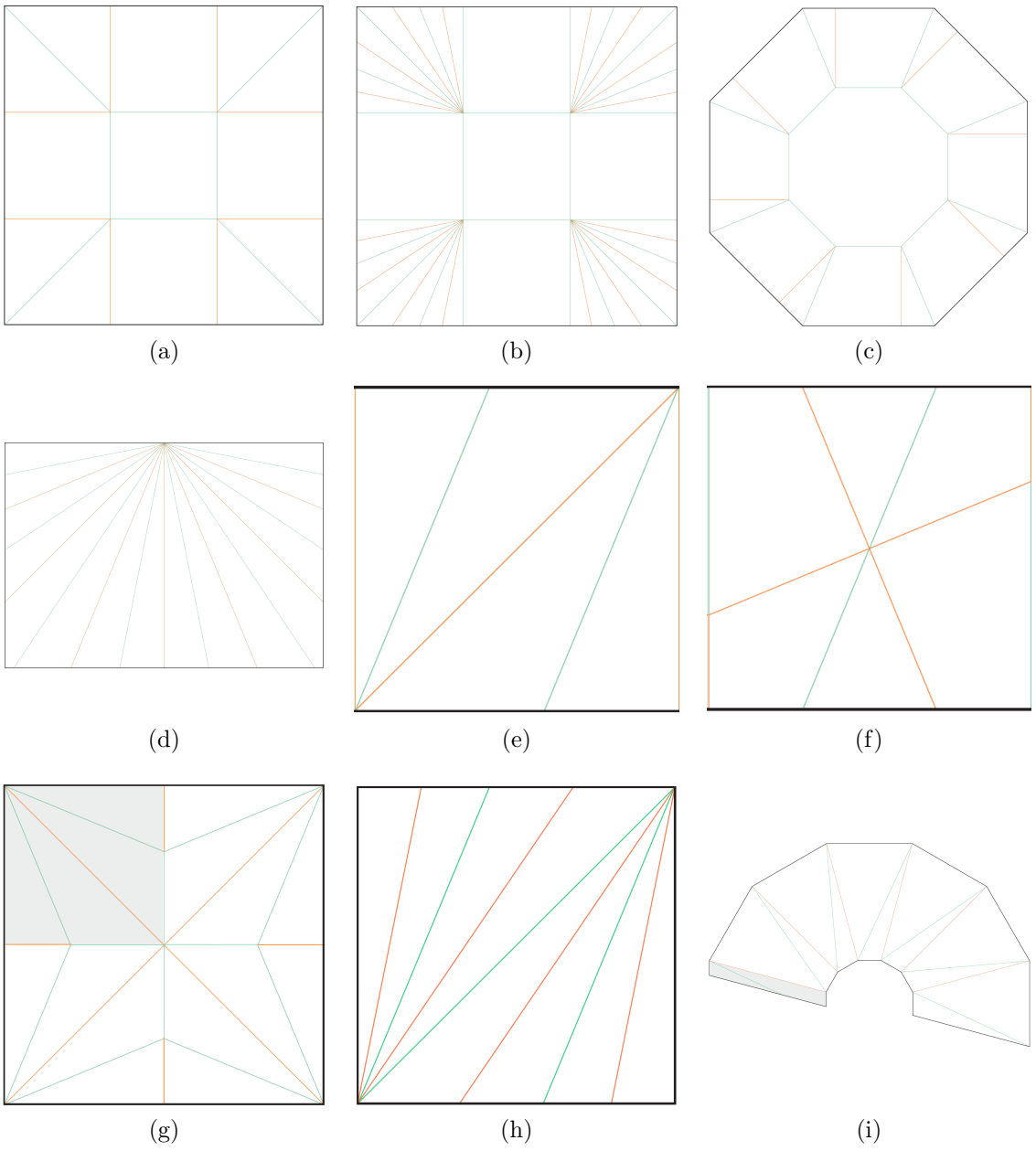


Figure 7.9: Examples of the crease patterns that cannot be made using either *OriPatch* or *Shady*. These patterns are composed of quadrilaterals and triangles that neither *OriPatch* nor *Shady* can handle. To manage to create these patterns, we should bring a new feature that allows a user to flexibly draw new edges to split faces. Therefore a user can select any two points along two edges of a face. This creates a straight line which connects the two points, which serves as a new edge to separate the face into two halves. Besides we should have a delete edge feature to remove any unwanted edges from a face. These cases involve some random geometric shapes and there are no clear patterns that can be simply generalized. For example, pattern (f) is formed by non-uniform quadrilaterals and triangles and is problematic to be generated procedurally.

REFERENCES

[1] Akitaya, H., Mitani, J., Kanamori, Y., and Fukui, Y. Origami diagrams and 3d animation from flat-foldable crease patterns sequences. In *SIGGRAPH 2013 poster* (2013).

[2] Alperin, R. C., and Lang, R. J. One-, two, and multi-fold origami axioms. *Origami 4* (2006), 371–393.

[3] Bern, M., and Hayes, B. The complexity of flat origami. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms* (1996), Society for Industrial and Applied Mathematics, pp. 175–183.

[4] Chang, S. H.-H., Stuart, L., Plimmer, B., and Wünsche, B. Origami simulator: A multi-touch experience. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems* (New York, NY, USA, 2009), CHI EA '09, ACM, pp. 3889–3894.

[5] Demaine, E., and Tachi, T. *Origamizer: A practical algorithm for folding any polyhedron.* Manuscript, 2010.

[6] Demaine, E. D., and ORourke, J. *Geometric folding algorithms*. Cambridge University Press Cambridge, 2007.

[7] Furuta, Y., Mitani, J., and Fukui, Y. A rendering method for 3d origami models using face overlapping relations. In *Smart Graphics* (2009), Springer, pp. 193–202.

[8] Greenberg, H., Gong, M., Magleby, S., and Howell, L. Identifying links between origami and compliant mechanisms. *Mech. Sci 2*, 2 (2011), 217–225.

[9] Hatori, K. K's origami: Origami construction. http://origami.ousaan.com/library/conste.html, 2006 (accessed June 17, 2014).

[10] Huang, Y., and Eisenberg, M. Easigami: Virtual creation by physical folding. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction* (New York, NY, USA, 2012), TEI '12, ACM, pp. 41–48.

[11] Huang, Y., Gross, M. D., Do, E. Y.-L., and Eisenberg, M. Easigami: A reconfigurable folded-sheet tui. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction* (New York, NY, USA, 2009), TEI '09, ACM, pp. 107–112.

[12] Huzita, H. Axiomatic development of origami geometry. In *Proceedings of the First International Meeting of Origami Science and Technology* (1989), pp. 143–158.

[13] J. Gout, X. Fouchet, V. O. Doodle. http://doodle.sourceforge.net/, (accessed May 21, 2014).

[14] Jackson, P. *Folding techniques for designers: from sheet to form*. Laurence King Publishing, 2011.

[15] Ju, W., Bonanni, L., Fletcher, R., Hurwitz, R., Judd, T., Post, R., Reynolds, M., and Yoon, J. Origami desk: Integrating technological innovation and human-centric design. In *Proceedings of the 4th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques* (New York, NY, USA, 2002), DIS '02, ACM, pp. 399–405.

[16] Justin, J. Resolution par le pliage de lequation du troisieme degre et applications geometriques. In *Proceedings of the First International Meeting of Origami Science and Technology* (1989), pp. 251–261.

[17] Kanade, T. A theory of origami world. *Artificial Intelligence 13*, 3 (1980), 279–311.

[18] Kilian, M., Flöry, S., Chen, Z., Mitra, N. J., Sheffer, A., and Pottmann, H. Curved folding. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 75:1–75:9.

[19] Lang, R. J. A computational algorithm for origami design. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry* (New York, NY, USA, 1996), SCG '96, ACM, pp. 98–105.

[20] Lang, R. J. Treemaker 4.0: A program for origami design. http://www.langorigami.com/science/computational/treemaker/TreeMkr40.pdf, 1998 (accessed June 18, 2014).

[21] Lang, R. J., and Hull, T. C. Origami design secrets: mathematical methods for an ancient art. *The Mathematical Intelligencer 27*, 2 (2005), 92–95.

[22] Lee, D.-Y., Kim, J.-S., Kim, S.-R., Koh, J.-S., and Cho, K.-J. The deformable wheel robot using magic-ball origami structure. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference IDETC/CIE* (Portland, Oregon, USA, 2013).

[23] Mitani, J. The folded shape restoration and the rendering method of origami from the crease pattern. In *13th International Conference on Geometry and Graphics* (Dresden, Germany, 2008).

[24] Mitani, J. Rendering method for flat origami. In *Eurographics'08: Annex to the Conference Proceedings* (Crete, Greece, 2008), pp. 291–294.

[25] Mitani, J. A design method for 3d origami based on rotational sweep. *Computer-Aided Design and Applications 6*, 1 (2009), 69–79.

[26] Mitani, J., and Igarashi, T. Interactive design of planar curved folding by reflection. In *19th Pacific Conference on Computer Graphics and Applications (Pacific Graphics 2011)* (Kaohsiung, Taiwan, 2011), pp. 77–81.

[27] Tachi, T. Generalization of rigid foldable quadrilateral mesh origami. In *Symposium of the International Association for Shell and Spatial Structures (50th. 2009. Valencia). Evolution and Trends in Design, Analysis and Construction of Shell and Spatial Structures: Proceedings* (2009), Editorial Universitat Politècnica de València.

[28] Tachi, T. Simulation of rigid origami. *Origami 4* (2009), 175–187.

[29] Tachi, T. Freeform rigid-foldable structure using bidirectionally flat-foldable planar quadrilateral mesh. *Advances in Architectural Geometry 2010* (2010), 87–102.

[30] Zhu, K., Deshan, C., and Fernando, O. N. N. Snap-n-fold: Origami pattern generation based real-life object structure. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems* (New York, NY, USA, 2012), CHI EA '12, ACM, pp. 2345–2350.