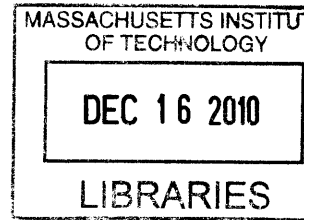


# Origami Transformers: Folding Orthogonal Structures from Universal Hinge Patterns

by

Aviv Ovadya

B.S., Massachusetts Institute of Technology (2009)



Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

**ARCHIVES**

Master of Engineering in Electrical Engineering and Computer Science


at the

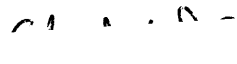
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

© Massachusetts Institute of Technology 2010. All rights reserved.

Author .....  
Department of Electrical Engineering and Computer Science  
August 20, 2010

Certified by .....  
  
Erik D. Demaine  
Associate Professor  
Thesis Supervisor

Accepted by .....  
  
Dr. Christopher J. Terman  
Chairman, Department Committee on Graduate Theses



# Origami Transformers: Folding Orthogonal Structures from Universal Hinge Patterns

by

Aviv Ovadya

Submitted to the Department of Electrical Engineering and Computer Science  
on August 20, 2010, in partial fulfillment of the  
requirements for the degree of  
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

We investigate a new approach to origami design using simple universal hinge patterns where the crease patterns for different shapes are just different subsets of a common hinge pattern. Several algorithms have previously been developed to design folded states for particular shapes, but they require a different crease pattern for each shape. Our motivations include the development of robotic “origami transformers,” artistic tools, and theoretical insights. We show how to compose “cube gadgets” to fold any  $N$ -cube polycube from an  $O(N) \times O(N)$  rectangle of paper, using only  $O(N^2)$  time to compute the parameters of the unambiguous folding sequence. We also describe extensions of our basic algorithm to larger classes of shapes with improved paper efficiency. Finally, we demonstrate that an implementation of this technique can actually be used to partially automate geometric paper folding.

Thesis Supervisor: Erik D. Demaine  
Title: Associate Professor



## Acknowledgments

I would like to express my gratitude to my supervisor Prof. Erik Demaine who was all around amazing, and who gave me the opportunity to merge my interests of computer science and origami. I would also like to thank my family for supporting and encouraging me through this entire process, from birth to thesis.

I would particularly like to acknowledge Simone Agha, Clare Bayley, Tucker Chan, Martin Demaine, Kimberley Dietz, Lyla Fischer, Andrew Geng, Jason Gross, Andrea Hawksley, Robert Johnson, Scott Johnson, Vincent Lee, Scott Macri, Maria Monks, and Xiaoyue Zhang for being sounding boards, providing equipment, folding, and/or proofreading for this research. I would also like to thank and acknowledge the following people for contributing figures or parts of figures: Scott Macri (Figure 1-3), Jason Ku (Figure 1-2; folded states), and Tiffany Tseng (Figures 3-2, 3-3, and 3-4; folded states).



# Contents

<b>List of Figures</b>	<b>9</b>
<b>0 Introduction</b>	<b>13</b>
0.1 Design Algorithms . . . . .	13
0.2 Motivation . . . . .	14
0.3 Results . . . . .	15
<b>1 Universality</b>	<b>19</b>
1.1 Definitions . . . . .	19
1.2 Cube Gadgets . . . . .	22
1.3 Folding Polycubes . . . . .	27
1.3.1 Hinge Pattern Completeness . . . . .	29
1.3.2 Paper Dimensions . . . . .	30
1.3.3 Number of Layers . . . . .	31
<b>2 Data Structures</b>	<b>33</b>
2.1 Polycube Representation . . . . .	33
2.1.1 Polycube Coordinate System . . . . .	34
2.1.2 Extrusion Sequence . . . . .	34
2.1.3 Cube Orientations . . . . .	34
2.2 Paper Allocation . . . . .	35
2.2.1 Face Grid . . . . .	35
2.2.2 Generating the Face Grid Sequence . . . . .	36

2.2.3	Cube Gadget Sequences . . . . .	37
2.3	Folding Representation . . . . .	38
2.3.1	Fold Pattern Sequences . . . . .	38
2.3.2	Complete Fold Pattern . . . . .	39
2.4	Performance . . . . .	40
<b>3</b>	<b>Extensions</b>	<b>41</b>
3.1	Naïve Surface Extrusions . . . . .	42
3.1.1	Polygrove Representation . . . . .	42
3.1.2	Face Grid Base Case . . . . .	43
3.2	Pleat-Sharing Surface Extrusions . . . . .	43
3.2.1	Extrusion Strategy . . . . .	43
3.2.2	Pleat Allocation . . . . .	44
3.2.3	Generating Face Grids . . . . .	45
3.2.4	Generating Cube-Gadget Steps . . . . .	46
3.2.5	Generating Pleat-Sharing Fold Pattern . . . . .	46
3.3	Side-Sharing Surface Extrusions . . . . .	46
3.3.1	Cube Gadget Wings . . . . .	47
3.3.2	Side-Sharing Strategy . . . . .	48
3.3.3	Generating Side-Sharing Cube-Gadget Steps . . . . .	49
3.3.4	Generating Side-Sharing Fold Pattern . . . . .	49
3.4	Performance . . . . .	49
3.4.1	Paper Size . . . . .	49
3.4.2	Run-time . . . . .	50
<b>4</b>	<b>Implementation</b>	<b>53</b>
4.1	Ruby Implementation . . . . .	53
4.2	Partial Folding Automation . . . . .	54
	<b>Bibliography</b>	<b>57</b>
	<b>Index</b>	<b>59</b>



# List of Figures

0-1	Origami transformer implementation described in “Programmable matter by folding” [6]. Select frames are shown from a video of a single actuated hinge pattern self-folding into a boat (left) and a plane (right). Photos by the Harvard Microrobotics Lab. . . . .	14
0-2	Folding a bend-shaped polycube with a square base via a CEA. For simplicity, the mountain-valley pattern in this and other figures does not exactly show the reflected creases when multiple layers are folded.	15
0-3	The mountain-valley pattern (left) and folded state (right) of a polycube representation of a car. . . . .	16
1-1	Abstract effect of applying a $[1, 2]$ -cube gadget at square $s_{4,5}$ of a $6 \times 7$ rectangle of paper. The two leftmost diagrams are top views before and after folding. The right diagram is a stylized perspective view of the folded state. . . . .	23
1-2	The hinge patterns (top), mountain-valley patterns (middle), and semi-transparent folded states (bottom) for the three cube gadgets. The highlighted region of each mountain-valley pattern has dimensions $2r + 1 \times 2c + 1$ and is the region used to actually fold the cube (half-square pleats extend out from those regions). . . . .	24
1-3	Given a piece of paper with a cube already folded on it, this diagram shows in an abstract manner how the paper moves when a new cube is folded depending on whether the old cube is on the top face (above), or a side face (below) of the new cube. . . . .	25

1-4	Given a coalesce sequence $C^P$ , shows the application of Lemma 1 to generate $C^{P'}$ with an additional cube. The purple regions show where additional rows and columns are inserted. We use the $\arctan \frac{1}{2}$ cube gadget in all figures from here onwards for consistency. . . . .	26
1-5	This sequence of folded states for a particular coalesce sequence shows an example of self intersection in part of a coalesce folding sequence. The self intersection occurs at the highlighted cube and is resolved in the final step. The self intersection can be avoided in this case by making additional simple folds. . . . .	27
1-6	This horizontal L-shaped polycube uses $\Omega(N^2)$ layers when folded by the Cube Extrusion Algorithm. . . . .	32
3-1	An example of a polygrove — one or more polycubes resting at integral points on a rectangle in the $xy$ plane. . . . .	41
3-2	An example of how the incremental improvements to CEA-2 affect the mountain-valley pattern, rectangle size, and folded state of a simple two cube polygrove. . . . .	42
3-3	The mountain-valley pattern and folded form associated with the pleat-sharing example described in Section 3.2. Note that the highlighted rows are shared by two cubes. . . . .	44
3-4	The mountain-valley pattern and folded form associated with the side-sharing example described in Section 3.3. Note that the highlighted area denotes where columns are removed due to sides shared by the two cubes. . . . .	47
4-1	The process by which paper origami can be constructed using the described algorithms. . . . .	53

4-2 A display at the Origami Museum at Narita Airport, Tokyo. Left: “Cube Evolution,” a succession of three simple polycubes from back to front, each with one additional cube (requires CEA-3.1). Back: “Inverse Cube Ring” which was designed with a single extrusion step with shared pleats and turned inside out (requires CEA-3.2). Right: “Low Resolution Car” which was designed with three extrusion steps with shared sides and pleats (requires CEA-3.3). . . . . 55



# Chapter 0

## Introduction

In this thesis, we describe the first steps toward a theoretical model of universal origami transformers. An “origami transformer” is a programmable surface made of flat plates connected by actuated hinges that can fold itself into more than one different shape. By “universal” we mean that the transformer can fold to match any shape up to a desired resolution. Finally, “theoretical model” implies that what we describe is not necessarily practical, but is both mathematically grounded, and useful for understanding the potential and limitations of universal origami transformers.

### 0.1 Design Algorithms

This work is in the field of computational origami (a subfield of computational geometry), and is specifically focused on design algorithms. There is a wide variety of previous research in this area over the past two decades, and also a strong artistic tradition. Techniques to fold flat appendage-based structures, which can then be shaped into representational origami such as animals, have been developed by many origami artists. Robert Lang’s *Origami Design Secrets* [7] is a practical guide to many of these techniques and the algorithms underlying some of these techniques have been explored by Lang and various Japanese mathematicians.

Research in the field accelerated in 1995 when Bern and Hayes [1] showed that folding was “hard” and therefore interesting to computer scientists [8]. One early

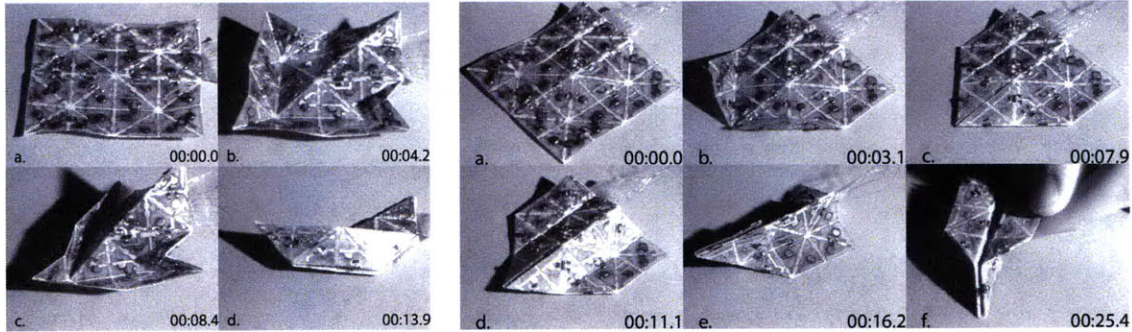


Figure 0-1: Origami transformer implementation described in “Programmable matter by folding” [6]. Select frames are shown from a video of a single actuated hinge pattern self-folding into a boat (left) and a plane (right). Photos by the Harvard Microrobotics Lab.

result is that every polyhedral surface can be folded from a large enough square of paper [2]. This result is broad but does not generate foldings that fulfill any intuitive notion of stability — they are wrappings of the surface by a long narrow strip. A more recent algorithm by Tomohiro Tachi attains much more “structural” foldings [5]. However, each polyhedral surface to be folded has a completely different crease pattern. So if we were to design a general “hinge pattern” on a piece of paper, some subset which is the crease pattern for a particular shape, such an algorithm would need a new hinge pattern for every shape. We focus here on exploring algorithms that work with “universal hinge patterns” for which different crease-pattern subsets fold into some universal class of shapes (up to a desired resolution).

## 0.2 Motivation

We have three primary motivations: robotics, art, and theory. Our robotics or “transformers” motivation is that we hope to develop “programmable matter” out of a foldable sheet [6]. The idea is to statically manufacture a sheet with specific actuated hinges that can self fold in either direction, and then dynamically program the fold angle of each hinge in the sheet (Figure 0-1). Thus a single manufactured sheet can be programmed to fold into anything that the hinge pattern can fold.

Our artistic motivation is that we want to fold geometric, perplexing, and beautiful

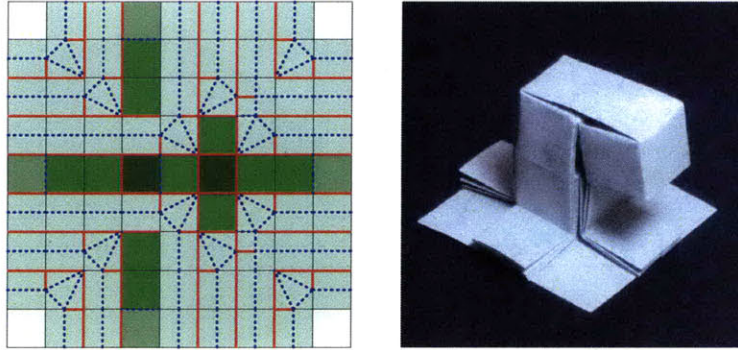


Figure 0-2: Folding a bend-shaped polycube with a square base via a CEA. For simplicity, the mountain-valley pattern in this and other figures does not exactly show the reflected creases when multiple layers are folded.

shapes out of paper. The folded “bend” in Figure 0-2 developed by our algorithms satisfies at least some of these criteria as it is obviously geometric, and it is rather perplexing to some that such a thing could be folded from a single uncut square. Other designs generated by the algorithms we describe might impress less geometrically inclined observers, especially with slight modifications for artistic effect.

Finally, we are motivated by a “desire to understand” — by theory. We would like to know what is possible and what is not possible. We want to know how to construct anything and how efficient those constructions are, in both the size of the paper required and the time required for the design computation.

### 0.3 Results

Our main result is that an  $O(N) \times O(N)$  square tiling of a simple hinge pattern can fold into all face-to-face gluings of  $N$  unit cubes (polycubes), and we describe a family of Cube Extrusion Algorithms which produce these foldings. Thus, by setting the resolution  $N$  sufficiently large, we can fold any 3D solid up to a desired accuracy.

At the core of our algorithm is the notion of a *cube gadget*, which folds a cube in the middle of a sheet of paper. Such foldings of a single cube have been independently developed by many origamists over the years; the first documented design we are

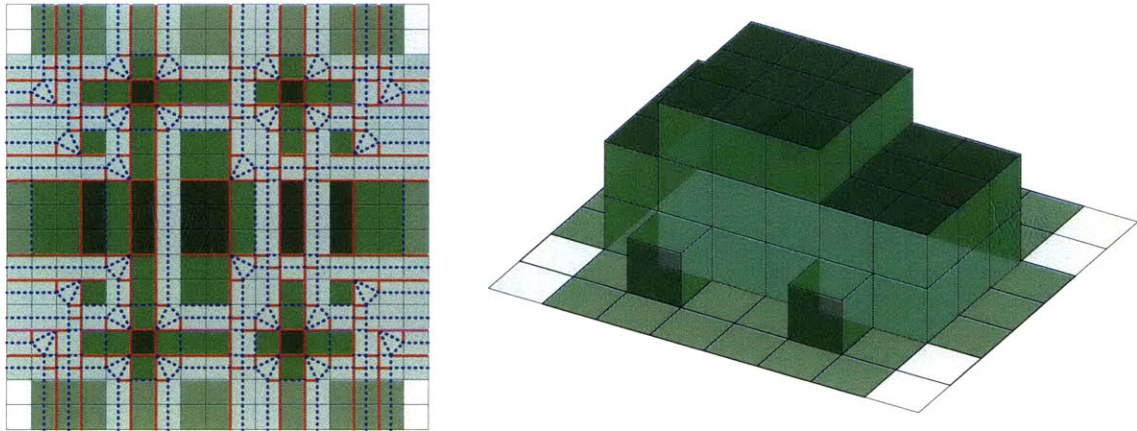


Figure 0-3: The mountain-valley pattern (left) and folded state (right) of a polycube representation of a car.

aware of was created by David A. Huffman in 1978.<sup>1</sup> The novelty is the way in which we compose and modify cube gadgets to form a desired polycube. We present three different cube gadgets, one of which is the gadget independently created by Huffman, each with its own advantages and disadvantages when combined to fold a polycube. (Only a single kind of cube gadget is used in a particular folding, but it is used  $N$  times.)

We first describe the algorithm geometrically in Chapter 1, and then we give a more detailed description based on data structures in Chapter 2. We generalize and improve the algorithm in Chapter 3, showing that in some cases we can do with as little as  $O(\sqrt[3]{N}) \times O(\sqrt[3]{N})$  paper (though  $\Theta(N) \times \Theta(N)$  is optimal in the worst case). Finally, in Chapter 4 we describe an implementation of the algorithm which can be used to automate experimentation and design of geometric origami using a cutting plotter or laser cutter to score the paper.

Figure 0-3 shows an example of a real origami design computed with our implementation, which also illustrates the improved efficiencies of the algorithms described in Chapter 3. This design is the basis for the “Low Resolution Car,” part of the exhibit shown in Figure 4-2.

---

<sup>1</sup>Personal communication with the Huffman family, 2010. The author independently developed this gadget in middle school circa 2000, and published an origami model based off it in 2004.



Throughout this thesis, we focus on construction, not analysis, and on algorithms with “sufficient” but not optimal foldings, as our model of folding is too simplistic to capture true optimality or “niceness” of a folded state. However, we do observe that the family of Cube Extrusion Algorithms are occasionally optimal even with the constant factors, and we make several conjectures about when this may be true. Our main contribution is showing how a single relatively simple gadget allows us to fold any shape, and we then take advantage of opportunities to remove wasted paper while maintaining the underlying structure of the gadget. The use of additional gadgets does allow greater flexibility and efficiency, but that is not the focus of this thesis.

Throughout this thesis you will see terms emphasized with bold italics as they are defined. The page number of each such definition appears in the index at the end of this thesis (and in the electronic version page numbers in the index hyperlink to the corresponding page). In addition, the index provides the page number for the introduction of notation if that notation is used consistently throughout the thesis.



# Chapter 1

## Universality

This chapter describes the basic Cube Extrusion Algorithm (**CEA-1**) and explores some properties of the folded states it generates. It takes a point-by-point geometric approach to describing paper and folding as opposed to the data-structure approach in later chapters.

### 1.1 Definitions

We start with a few definitions about origami, specified somewhat informally for brevity. For more formal definitions, see [4, ch. 11].

For our purposes, a *piece of paper* is a connected collection of flat polygons in 3D joined along shared edges (a polyhedral complex; note that we can have multiple polygons in the same place but with different connections, and with a specified stacking order). A notable special case is a single  $m \times n$  rectangle of paper for integers  $m$  and  $n$  (but in general, a piece of paper does not have to be flat; for example, a polyhedron is a piece of paper). We index the unit squares of such a rectangle in the style of matrices:  $s_{i,j}$  refers to the unit square in the  $i$ th row and  $j$ th column, and  $s_{1,1}$  is in the upper-left corner.

A *hinge* is a line segment drawn on a piece of paper which is capable of being creased in either direction. A *hinge pattern* is a collection of hinges drawn on a piece of paper. The hinge patterns we consider in this paper are all based on subdivisions

of the unit-square grid, adding a finite number of hinges within each unit square. The unit squares of the hinge pattern correspond to the unit squares of a rectangle of paper.

An example of a hinge pattern is the *box-pleated pattern* (known in geometry as the *tetrakis tiling*) which is formed from the unit-square grid by subdividing each square in half vertically, horizontally, and by the two diagonals, forming eight right isosceles triangles. The upper-left corner of Figure 1-2 shows an example for four unit-squares.

An *angle pattern* is a hinge pattern together with an assignment of a real number in  $[-180^\circ, +180^\circ]$  to each hinge, specifying a fold angle (negative for valley, positive for mountain). We allow a hinge to be assigned an angle of 0, in which case we call the hinge *trivial*, though we do not draw trivial hinges in most figures. A hinge with a nonzero angle is called a *crease*. The *crease pattern* is the subgraph of the hinge pattern consisting of only the creases.

An angle pattern determines a 3D geometry called the *folded geometry*, which maps each face of the crease pattern to a 3D polygon via a Euclidean isometry (by the composition of rotations at creases). More explicitly, a folded geometry is a map from all points of the piece of paper to  $\mathbb{R}^3$  that satisfies constraints as specified in [4, ch. 11]—and there is an obvious mapping from angle patterns to folded geometries.

A *folded state* consists of such a folded geometry together with an ordering  $\lambda$ , which is a partial function over the touching points in the folded geometry, in our case describing the stacking relationship among polygons of the crease pattern that touch in the folded geometry. Define the *starting sheet* of a folded state to be the original piece of paper, that is, the domain of the folded geometry.<sup>1</sup> A *folding sequence* is a sequence of folded states  $F_1, F_2, \dots, F_k$  from the same starting sheet. The last folded state in a folding sequence is called the *final folded state*.

Define the *number of layers at a point*  $q$  to be the number of noncrease points in the piece of paper that get mapped to  $q$  by the folded geometry. The *number of*

---

<sup>1</sup>Note that “sheet” is not to suggest that the piece of paper needs to be flat; it can be any polyhedral complex.

**layers of a folded state** is the maximum number of layers over all points.<sup>2</sup>

Next we define a notion of “coalescing” which lets us ignore certain details of a folded state. A **coalesce folded state** is a folded state augmented with a **coalesce set** which is a subset of the starting sheet. If we take the starting sheet and identify (glue together) all pairs of points in the coalesce set that are collocated by the folded geometry, then we obtain a metric space called the **coalesce result**. This coalesce result is also a piece of paper under our definition and therefore can be the domain of a new coalesce folded state. A **coalesce sequence** is a sequence  $C_1, C_2, \dots, C_k$  of coalesce folded states, where each  $C_k$  is a folding of the coalesce result of  $C_{k-1}$ .

One can generate a folding sequence from a coalesce sequence by letting  $F_1 = C_1$  and  $F_k = F_{k-1} \circ C_k$ , and then composing the geometry and ordering functions in the obvious way. Note that the starting sheet of each  $F_k$  is the starting sheet of  $C_1$ , while the starting sheets of the other  $C_k$ 's can be any shape folded from that starting sheet. The **final folded state of a coalesce sequence** is the final folded state of the generated folding sequence. We say that a folding sequence or coalesce sequence *folds a piece of paper  $\pi$  into a shape  $\sigma$*  if the starting sheet of the first folded state is the piece of paper  $\pi$  and the image of the last folded geometry is the shape  $\sigma$ .

In this paper we allow all but the last folded state in a folding sequence or coalesce sequence to have crossings. By [3], all folded states are reachable from the starting sheet by the continuous folding motion, so the final folded state is still reachable. We use folding sequences as a tool to construct the final folded state, not as instructions for folding.

Now that we can describe how to fold a shape, we define our target shapes. A **polycube**  $P$  is a union of unit cubes on the unit-cube lattice with a connected **dual graph**; the dual graph has a vertex for each unit cube and an edge between two vertices whose corresponding cubes share a face. The **faces** of the polycube are the (square) faces of the individual cubes that are not shared by any other cubes.

A **folding of a polycube** is a folded state that covers all faces of the polycube,

---

<sup>2</sup>This measure is a simple way to bound the effect of paper thickness, but in practical origami there are other quantities that could be measured.

and nothing outside the polycube. In fact, some of our foldings of polycubes will also include the internal squares, the faces shared by multiple cubes, and some of our foldings will not put anything else interior to cubes, but in general we do not require either property. A face of a folded polycube is *seamless* if the outermost layer of paper covering it is an uncreased unit square of paper. Our foldings will generally be seamless.

## 1.2 Cube Gadgets

We now introduce the notion of a cube gadget; refer to Figure 1-1. For positive integers  $r$  and  $c$ , an  $[r, c]$ -**cube gadget** is a method of extruding a cube from a rectangular piece of paper at a specified location. The input to the cube gadget is an  $m \times n$  rectangle of paper, for integers  $m > 2r$  and  $n > 2c$ , as well as a unit square  $s_{i,j}$  on the paper, where  $r < i < m - r$  and  $c < j < n - c$ . The output of the cube gadget is a folding of the  $m \times n$  rectangle into the shape of a cube sitting on a smaller,  $(m - 2r) \times (n - 2c)$  rectangle of paper. The cube sits on the square  $s_{i-r, j-c}$  in the smaller sheet of paper. All six faces of the cube are seamless except for the bottom face. The top face of the cube is covered by square  $s_{i,j}$  from the original piece of paper. The boundary of the original  $m \times n$  rectangle paper is mapped onto the boundary of the smaller  $(m - 2r) \times (n - 2c)$  rectangle.

The cube gadgets in this paper achieve the folding by making horizontal pleats in the  $r$  rows above and below row  $i$ , and making vertical pleats in the  $c$  columns left and right of column  $j$ . The pleats are called *half-square pleats* because they are composed of unit squares folded in half. Each pleat adds two layers to the row or column it is under, so the number of layers of the folded state is at least  $1 + 2 \max\{r, c\}$  (one for the row or column plus two for each pleat).

Another property of our foldings is that all folding is within the  $2r + 1$  rows and  $2c + 1$  columns surrounding square  $s_{i,j}$ . Thus, the quadrant of paper consisting of rows  $< i - r$  and columns  $< j - c$  is not folded and is incident to the top-left corner of the cube, and similarly for the other four quadrants.

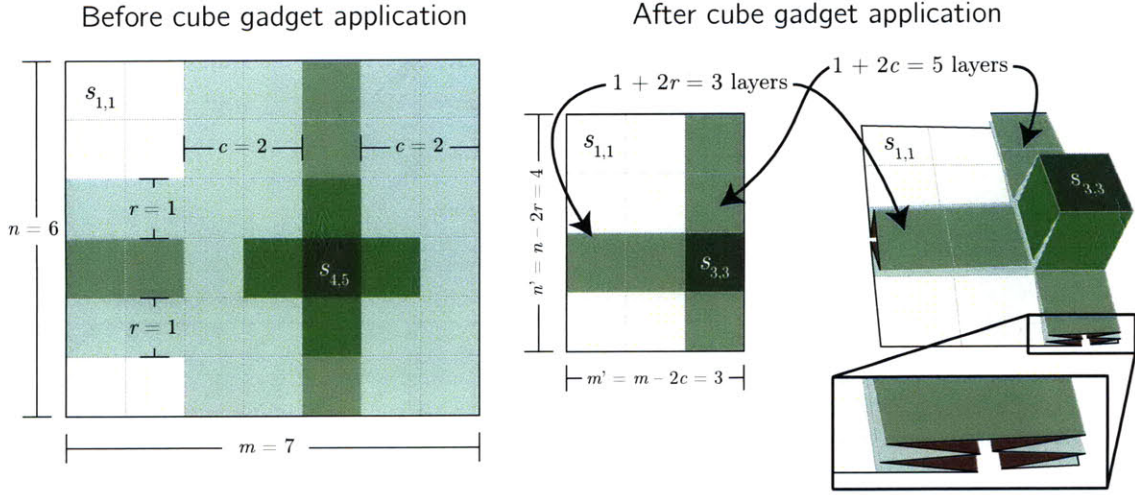


Figure 1-1: Abstract effect of applying a  $[1, 2]$ -cube gadget at square  $s_{4,5}$  of a  $6 \times 7$  rectangle of paper. The two leftmost diagrams are top views before and after folding. The right diagram is a stylized perspective view of the folded state.

In this paper, we give three different cube gadgets based on three different hinge patterns, as shown in Figure 1-2. The three cube gadgets are based, respectively, on the box-pleated pattern, the *slit pattern*, and the *arctan  $\frac{1}{2}$  pattern*. The *arctan  $\frac{1}{2}$  gadget* and *slit gadget* are  $[1, 1]$ -cube gadgets, while the *box-pleated gadget* is a  $[1, 2]$ -cube gadget. The advantage of the *box-pleated gadget* and *slit gadget* is that the hinge pattern is simpler: box pleating has all creases with angles at integer multiples of  $45^\circ$ . The slit gadget attains higher efficiency than seems possible with regular box pleating by adding a regular pattern of slits in the paper. The arctan  $\frac{1}{2}$  gadget attains higher efficiency using more hinges some of which are at angles of arctan  $\frac{1}{2}$ . We use the arctan  $\frac{1}{2}$  gadget in all figures for consistency.

Next, we show how a cube gadget can be used to modify an existing folding, which will be the key construction in our folding of general polycubes. Figure 1-3 provides some intuition for how existing cubes move as new cubes are folded and Figure 1-4 provides a formal example of the lemma below.

**Lemma 1 (Gadget Application)** *Let  $C^P$  be a coalesce sequence for a polycube  $P$  from an  $m \times n$  rectangle of paper. Let  $f$  be a face of the polycube  $P$  that is seamless in the final folded state of  $C^P$ . Then there is a coalesce sequence  $C^{P'}$  for the polycube  $P'$ ,*

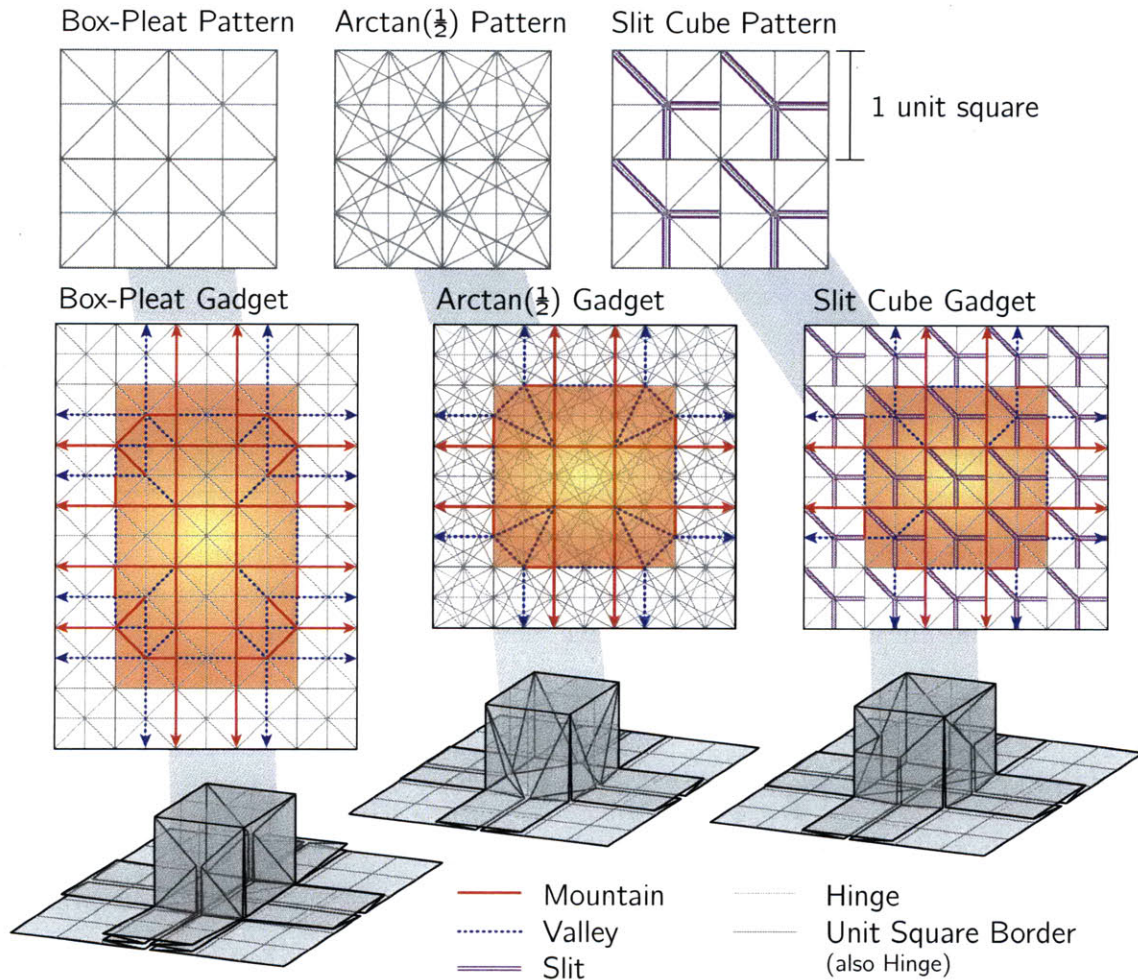


Figure 1-2: The hinge patterns (top), mountain-valley patterns (middle), and semi-transparent folded states (bottom) for the three cube gadgets. The highlighted region of each mountain-valley pattern has dimensions  $2r + 1 \times 2c + 1$  and is the region used to actually fold the cube (half-square pleats extend out from those regions).

consisting of  $P$  plus a cube extruded from face  $f$ , from an  $(m + 2r) \times (n + 2c)$  rectangle of paper. The construction is parameterized by a cube gadget.

**Proof:** Let  $C^P = C_1^P, C_2^P, \dots, C_q^P$ , and let  $F_q^P$  be the final folded state (for  $P$ ). Let  $s_{i,j}$  be the square in the starting sheet of  $F_q^P$  that is mapped to  $f$  via  $F_q^P$ . We use  $\sigma$  to refer to this square in an abstract sense—when we add rows or columns to the start sheet  $\sigma$  will move with the insertions—so the square coordinates referred to by  $\sigma$  may change.

We construct a new coalesce folded state  $C_1^{P'}$  that we will prepend to  $C^P$ . Define



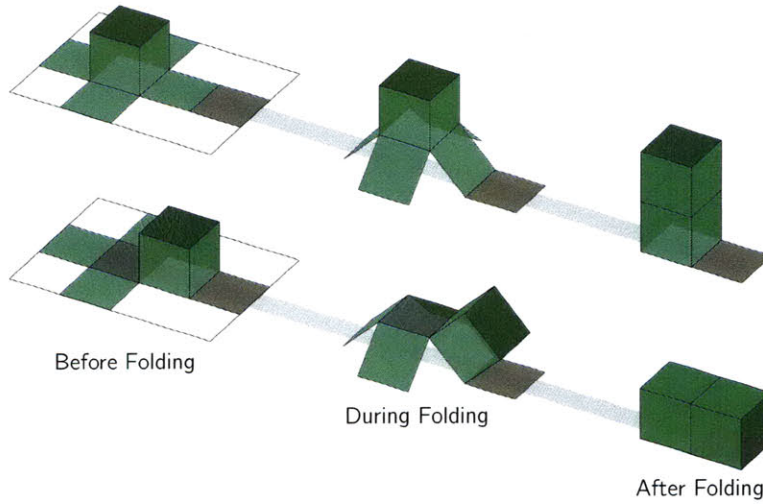


Figure 1-3: Given a piece of paper with a cube already folded on it, this diagram shows in an abstract manner how the paper moves when a new cube is folded depending on whether the old cube is on the top face (above), or a side face (below) of the new cube.

$C_1^{P'}$  to have the same starting sheet as that of  $F_q^P$ , except that we insert  $r$  rows above  $\sigma$ ,  $r$  rows below  $\sigma$ ,  $c$  columns left of  $\sigma$ , and  $c$  columns right of  $\sigma$ . So the starting sheet is a rectangle of paper of size  $(m + 2r) \times (n + 2c)$  and  $\sigma$  refers to the square  $s_{i+r,j+c}$  in this enlarged sheet of paper. Define this entire enlarged rectangle to be the coalesce set of  $C_1^{P'}$ . Now we define the folded state of  $C_1^{P'}$  to be the given cube gadget applied at  $\sigma$ . The result looks like a cube sitting on the square  $s_{i,j}$  of an  $m \times n$  rectangle of paper. (The paper does have additional layers in some places from the pleats, but it folds flat except at the cube, and these layers are all coalesced because the entire sheet is in the coalesce set.)

Now, for each coalesce folded state  $C_k^P$ , we create a modified coalesce folded state  $C_{k+1}^{P'}$  with  $\sigma$  replaced by a cube of paper. Here we use the fact that  $\sigma$  never gets folded throughout the sequence (since it is always the face of a cube), and thus corresponds to a seamless square of paper in the starting sheet of each coalesce folded state  $C_k^P$ . Note that we add the five new faces of the cube to the starting sheet, but we do not add these faces to the coalesce set of  $C_{k+1}^{P'}$ ; the latter will remain a rectangle. We also add any polygons of paper internal to the cube that appear in  $C_1^{P'}$ , in the same orientation. Because the coalesce result of  $C_k^P$  is the starting sheet of  $C_{k+1}^{P'}$ , we have

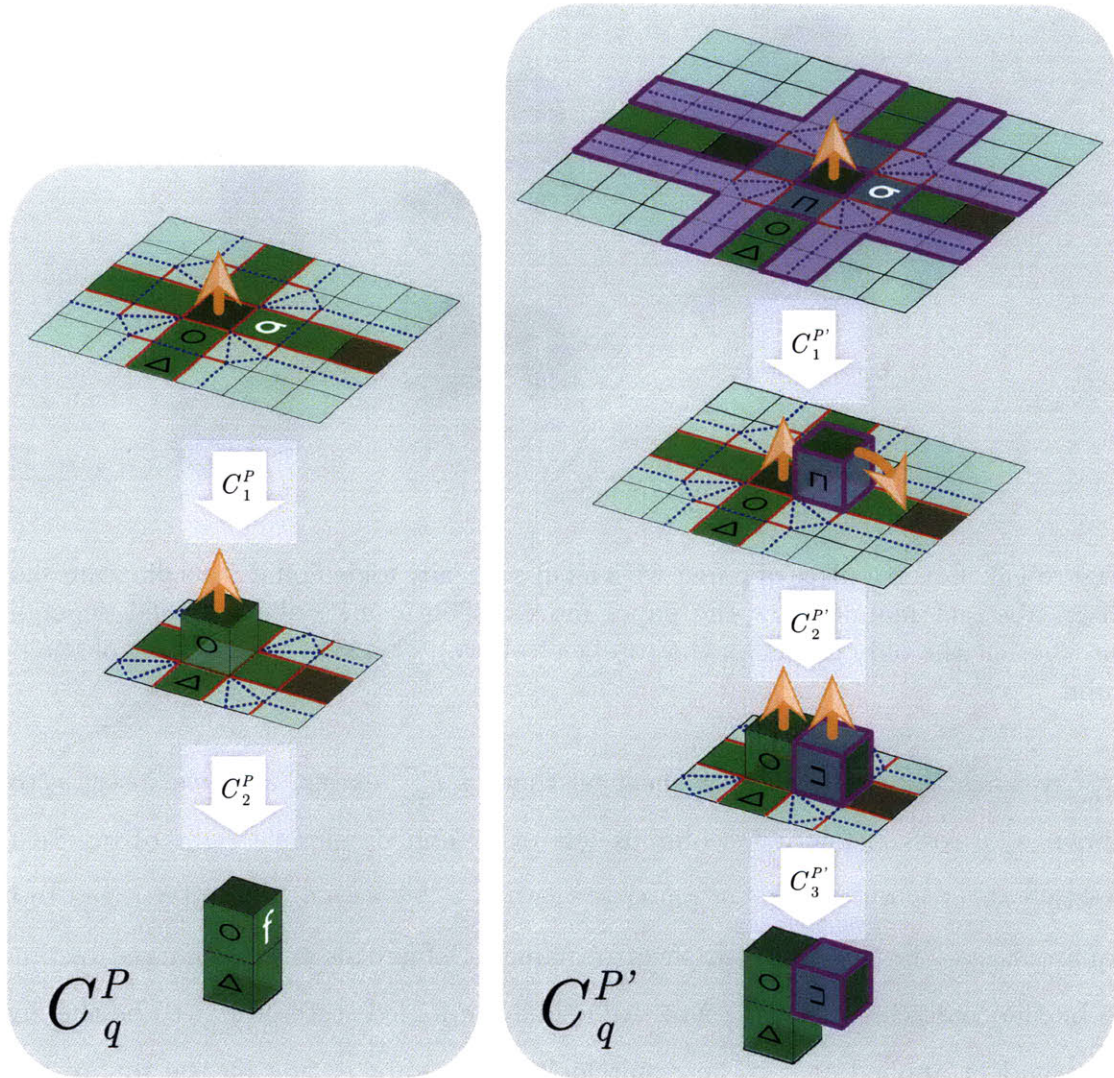


Figure 1-4: Given a coalesce sequence  $C^P$ , shows the application of Lemma 1 to generate  $C^{P'}$  with an additional cube. The purple regions show where additional rows and columns are inserted. We use the  $\arctan \frac{1}{2}$  cube gadget in all figures from here onwards for consistency.

thus generated a new coalesce sequence  $C^{P'} = C_1^{P'}, C_2^{P'}, \dots, C_q^{P'}, C_{q+1}^{P'}$ .

Note that these added cubes may create intersections in the coalesce folded states  $C_k^{P'}$  (as mentioned in Section 1.1). However, the final folded state  $F_{q+1}^{P'}$  of  $C^{P'}$  (as well as  $C_{q+1}^{P'}$  itself) is guaranteed not to have intersections. This follows because  $F_q^P$  had no self intersections, the application  $C_1^{P'}$  of the cube gadget has no self intersections, and adding the cube of paper to make  $P$  into  $P'$  cannot create intersections.

□

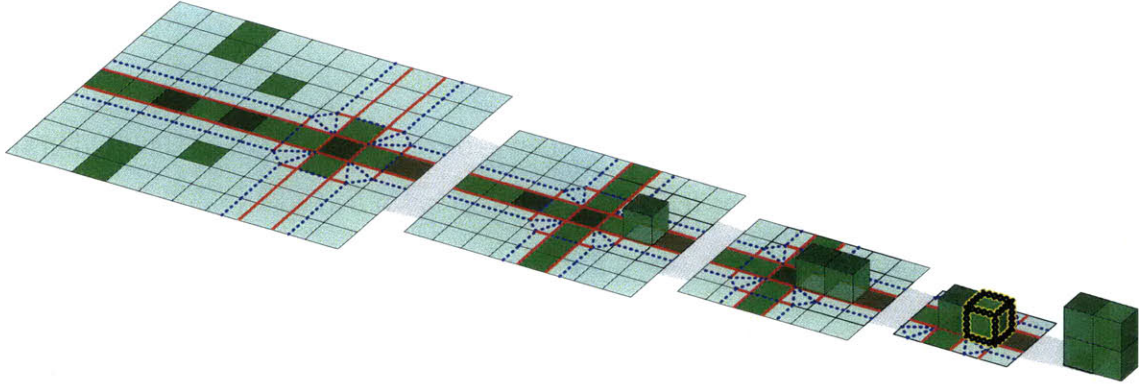


Figure 1-5: This sequence of folded states for a particular coalesce sequence shows an example of self intersection in part of a coalesce folding sequence. The self intersection occurs at the highlighted cube and is resolved in the final step. The self intersection can be avoided in this case by making additional simple folds.

### 1.3 Folding Polycubes

Our Cube Extrusion Algorithm (**CEA-1**) for folding any polycube is parametrized by an arbitrary cube gadget, and consists of repeated application of the gadget according to Lemma 1. We describe the recursive algorithm by way of an inductive proof:

**Theorem 2 (Cube Extrusion Algorithm)** *Any polycube of  $N$  cubes can be folded with all faces seamless from a  $(2rN + 1) \times (2cN + 2)$  rectangle of paper by a sequence of  $N$  applications of an  $[r, c]$ -cube gadget plus one additional fold.*

**Proof:** We prove by induction that any polycube  $P'$  of  $N$  cubes can be folded seamlessly by a coalesce sequence from a  $(2rN + 1) \times (2cN + 2)$  rectangle of paper. Arbitrarily choose a “bottom face”  $f_b$  of  $P'$ , and let  $b$  be the unique (bottom) cube having  $f_b$  as a face. Refer to Figure 1-4.

The base case is  $N = 1$ , when  $P'$  consists of the single cube  $b$ . We can use the cube gadget directly at square  $s_{r+1, c+1}$  to obtain a folding of the single cube from a  $(2r + 1) \times (2c + 2)$  rectangle of paper. The folded state is a cube next to a (pleated) unit square of paper. Note that the bottom face of the cube corresponds to  $f_b$ , and is adjacent to the square of paper. By definition of cube gadgets, all faces of the cube except the bottom face are seamless. We fold the extra square of paper over to

seamlessly cover the bottom face, thus making a seamless one-cube polycube. The resulting folded state forms the first and only step in a coalesce sequence.

It remains to prove the inductive step. Let  $T$  be a spanning tree of the dual graph of  $P'$ . Because every tree has at least two leaves,  $T$  has a leaf corresponding to a cube  $l \neq b$ . Let  $u$  be the unique cube sharing a face with  $l$ , and let  $f_{ul}$  be the face shared by  $u$  and  $l$ .

Now consider the polycube  $P = P' \setminus \{l\}$ , with  $N - 1$  cubes. Because  $l \neq b$ ,  $f_b$  remains a face of  $P$ . By induction, there is a coalesce sequence  $C^P$  that folds a  $(2r(N - 1) + 1) \times (2c(N - 1) + 2)$  rectangle of paper into  $P$ . By Lemma 1, we extrude from  $f_{ul}$  to obtain a new coalesce sequence  $C^{P'}$  for  $P'$  from a rectangle of size  $((2r(N - 1) + 1) + 2r) \times ((2c(N - 1) + 2) + 2c) = (2rN + 1) \times (2cN + 2)$ .

The final folded state of the inductively obtained coalesce sequence is the desired folded state from the rectangle of paper into the polycube.  $\square$

Without the concern for a seamless bottom face, we can reduce the  $+2$  in the rectangle bound down to  $+1$ .

The algorithm runs in polynomial time. The bottleneck is in converting the coalesce sequence into its final folded state. Each of the  $N$  cube gadgets causes the creation of at most  $O(N)$  creases, because the piece of paper at that point has size  $O(N) \times O(N)$  with  $O(1)$  existing creases per square.

We conjecture that the exact bound of a  $(2rN + 1) \times (2cN + 2)$  rectangle of paper is optimal for any sufficiently “zig zaggy” tree polycube (due to the way “wasted paper” is introduced and how the improvements described later can minimize it).<sup>3</sup> One fact along these lines is that the size bound of an  $O(N) \times O(N)$  rectangle of paper is tight up to constant factors for square paper. Specifically, folding a  $1 \times 1 \times N$  tower of cubes requires starting from a square of side length  $N$  in order to have diameter  $N$ , as folding can only decrease diameter.

---

<sup>3</sup>Part of the difficulty in saying something definite about optimality is that we can “move around pleats” with additional gadgets to achieve better foldings in many cases.

### 1.3.1 Hinge Pattern Completeness

Next we show that the Cube Extrusion Algorithm does not create creases that stray from the given hinge pattern.

For a rectangular piece of paper, the **tile**  $t_{i,j}$  of a crease pattern is the set of creases within the unit square  $s_{i,j}$ . The **tile set** of a cube gadget is the set of all distinct tiles that can be generated by the cube gadget. The hinge pattern **generated** by a tile is the result of replicating the tile in a unit-square grid.

**Proposition 3** *Given a cube gadget with a finite tile set and half-square pleats as the only folded structure outside of the cube, if we add to an empty tile a hinge for every crease of each tile in the tile set for every 2D orthogonal orientation (rotations and reflections), then the resulting tile generates the hinge pattern required to fold a polycube with the Cube Extrusion Algorithm.*

This proposition is nontrivial as it is possible that some combination of cube gadgets would create new tiles that are not present in any single gadget which are thus not in the tile set.

**Proof:** We prove that no other hinges are needed beyond those found in the constructed generator tile.

There are two types of folded tiles used by cube gadgets to make polycubes as described in the proposition: **inner tiles** which make up the non-visible parts inside the cubes and **pleat tiles** that make up the non-visible part of the pleats (outside of the cube). Inner tiles are never folded once they are made part of a cube as our foldings never modify the inner structure of an existing cube, so they do not require any additional hinges beyond those in the tile set. Pleat tiles may be folded again — but they are all half-square pleats, which means that they simply reflect a crease along the midline of the tile — but each of the reflected halves would already have existed in reflections of that tile of the cube gadget, so this also does not create additional hinges. □

### 1.3.2 Paper Dimensions

We now show the specific bounds on the dimensions of the required rectangle of paper for each of the three cube gadgets considered in this paper.

**Corollary 4 (arctan  $\frac{1}{2}$  Universality Lemma)** *Any polycube of  $N$  cubes can be folded with all faces seamless from an arctan  $\frac{1}{2}$  hinge pattern on a  $(2N + 1) \times (2N + 2)$  rectangle of paper using the Cube Extrusion Algorithm.*

**Proof:** The arctan  $\frac{1}{2}$  gadget has  $r = 1$  and  $c = 1$ . Plugging these constants into Theorem 2 yields a process for folding the polycube from a rectangle of paper of size  $(2N + 1) \times (2N + 2)$ .  $\square$

**Corollary 5 (Slit Universality Lemma)** *Any polycube of  $N$  cubes can be folded with all faces seamless from a slit hinge pattern on a  $(2N + 1) \times (2N + 2)$  rectangle of paper using the Cube Extrusion Algorithm.*

**Proof:** Same as Corollary 4.  $\square$

The previously discussed gadgets create foldings from a rectangle of paper that is within an additive constant of being square. As we show now, directly applying the Cube Extrusion Algorithm with the tetrakis cube gadget generates a folding with a ratio within a constant of  $1 \times 2$ , but a slight modification allows us to use an approximately square sheet of paper.

**Corollary 6 (Box-Pleated Universality Lemma)** *Any polycube  $P$  of  $N$  cubes can be folded with all faces seamless from a box-pleated (tetrakis) hinge pattern on a  $(2N + 1) \times (4N + 2)$  rectangle of paper using the Cube Extrusion Algorithm. A slight modification of the Cube Extrusion Algorithm uses a  $(3N + 1) \times (3N + 2)$  rectangle of paper for even  $N$  and a  $(3N) \times (3N + 3)$  rectangle of paper for odd  $N$ .*

**Proof:** The box-pleated gadget has  $r = 1$  and  $c = 2$ . Plugging these constants into Theorem 2 yields a process for folding  $P$  from a rectangle of paper of size  $(2N + 1) \times (4N + 2)$ .

Now we describe the modified approach. Define the *transpose of a cube gadget* to be the cube gadget with  $r$  and  $c$  interchanged, so that now we insert  $r$  columns to the left and right of the column and  $c$  rows below and above the specified row. We alternate the box-pleated gadget and its transpose for a polycube of  $N$  cubes such that the box-pleated gadget is applied  $\lceil N/2 \rceil$  times and its transpose is applied  $\lfloor N/2 \rfloor$  times. This yields a final folded state  $F_N$  with a starting sheet of size  $(2r \cdot \lceil \frac{N}{2} \rceil + 2c \cdot \lfloor \frac{N}{2} \rfloor + 1) \times (2c \cdot \lceil \frac{N}{2} \rceil + 2r \cdot \lfloor \frac{N}{2} \rfloor + 2)$  which simplifies slightly to  $(2 \cdot \lceil \frac{N}{2} \rceil + 4 \cdot \lfloor \frac{N}{2} \rfloor + 1) \times (4 \cdot \lceil \frac{N}{2} \rceil + 2 \cdot \lfloor \frac{N}{2} \rfloor + 2)$ . For even  $N$ , this bound is  $(3N + 1) \times (3N + 2)$ , and for odd  $N$ , it is  $(3N) \times (3N + 3)$ .  $\square$

### 1.3.3 Number of Layers

**Proposition 7** *For any polycube of  $N$  cubes, the Cube Extrusion Algorithm produces a folding that uses  $O(N^2)$  layers.*

**Proof:** The folded state produced by Theorem 2 has a starting sheet of size  $(2rN + 1) \times (2cN + 2)$ , which is clearly  $O(N^2)$  for constants  $r$  and  $c$ . And because each square of a hinge pattern contains  $O(1)$  hinges (by definition), there can be at most  $O(N^2)$  layers in the folding (even if we folded the paper up into the smallest unit of area allowable by the hinge pattern).  $\square$

Unfortunately, this quadratic bound on the number of layers is tight in the worst case:

**Proposition 8** *For any  $N$  and any cube gadget  $G$ , there exists a polycube of  $N$  cubes for which the Cube Extrusion Algorithm yields a folding requiring  $\Omega(N^2)$  layers.*

**Proof:** Without loss of generality, assume that  $N$  is odd. The example we use is a horizontal L-shaped polycube, as shown in Figure 1-6. To construct it, take a single cube  $b$  and two faces which share an edge of the cube. Extrude  $(N - 1)/2$  cubes from each of the faces. (If  $N$  were even, we would extrude  $(N/2) - 1$  cubes from one of the faces and  $N/2$  cubes from the other face).

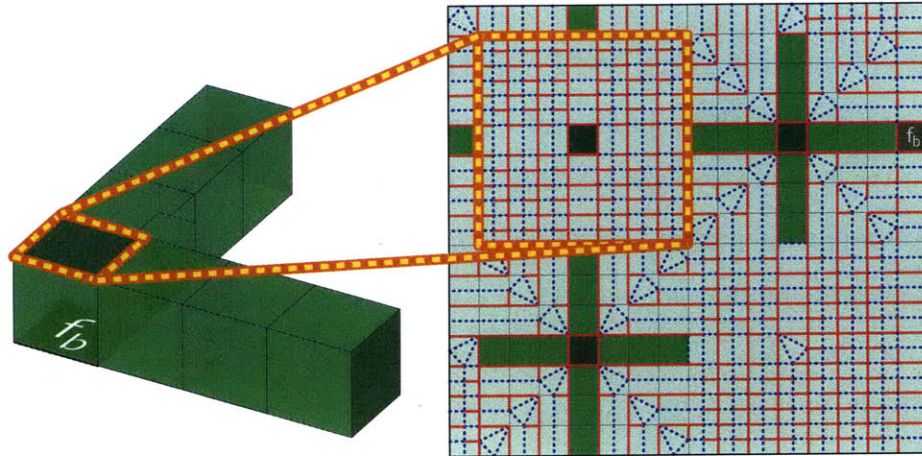


Figure 1-6: This horizontal L-shaped polycube uses  $\Omega(N^2)$  layers when folded by the Cube Extrusion Algorithm.

We now show that our folding algorithm would construct a folding having  $\Omega(N^2)$  layers. Let the bottom cube of the folding algorithm be  $b$ , and take the bottom face  $f_b$  to be one of the faces parallel the plane spanned by the legs of the L. Now consider the face in the resulting folded state opposite to  $f_b$ : it is seamless, but hidden beneath it are pleats from prisms of both legs of the L. There are  $\Omega(N)$  pleats from each leg, and the pleats are orthogonal to each other. This results in  $\Omega(N^2)$  layers.  $\square$



# Chapter 2

## Data Structures

The proof of universality in Chapter 1 provides a high-level sketch of an algorithm for designing a folding of a desired polycube. Implementation of such an algorithm on a real computer, or even a typical model of computation, requires substantially more detail, particularly in the data structures for representing the intermediate structures, geometries, and foldings. Here we provide these details for a directly implementable algorithm, which we denote *CEA-2*.

### 2.1 Polycube Representation

For the rest of this thesis, we need a more data structural perspective on the notion of a polycube. Namely, define a *polycube* to be a geometric graph with *vertices* embedded on the three-dimensional cube grid, where the *edges* are all grid edges. The vertices represent unit cubes, and the edges correspond to shared faces. There can be only one edge in each direction from a particular cube, so there is a maximum of six edges per vertex.<sup>1</sup> If an edge is not present we call it a *null edge* and we call the corresponding face *exposed*.

---

<sup>1</sup>There can however be multiple cubes in the same location, though this rarely comes up as it involves self intersection in the represented folded form, or cubes going below the root cube.

### 2.1.1 Polycube Coordinate System

Let  $P$  be a polycube with a **root cube**  $c_1$  that has an exposed face  $f_1$ . We describe a coordinate system that will be useful for manipulating  $P$ . The origin  $(0, 0, 0)$  is the center of the root cube. The inner normal of  $f_1$  points in the positive  $z$  direction. A perpendicular to a side of  $f_1$  in the  $xy$ -plane is designated as the positive  $x$  direction, and a perpendicular to a neighboring side of  $f_1$  in the  $xy$ -plane is designated as the positive  $y$  direction. We consider the **location** of a cube to be the coordinates of its center. Cubes are unit size, so all of the cubes of  $P$  will be located on integral points in this coordinate system. In general, we may refer to  $+x$  as **east** (e),  $-x$  as **west** (w),  $+y$  as **north** (n),  $-y$  as **south** (s),  $+z$  as **above** (a), and  $-z$  as **below** (b). This polycube-relative set of directions in the 3D orthogonal coordinate system is denoted as  $D_{PO}$  (**polycube orthogonal directions**).

### 2.1.2 Extrusion Sequence

Consider a spanning tree  $T$  of the polycube, rooted at  $c_1$ . Let the **extrusion sequence** be a sequence of cubes described by a pre-order traversal of  $T$  and let  $c_k$  be the  $k$ th cube in the sequence. So  $k$  from 1 through  $N$  is the **step number** and  $c_k$  is the cube that is “extruded” at **step**  $k$ . Let  $P_{1\dots k}$  be the sub-polycube with cubes from steps 1 through  $k$  (in this case  $c_1, \dots, c_k$ ). Note that  $P_{1\dots k}$  is a valid (connected) sub-polycube as the pre-order traversal ensures that a cube is not extruded unless its parent in the spanning tree had previously been extruded. Also note that these step indices are in the order of extrusion, which is the reverse order of folding (this is the opposite order of, for example, the coalesce folded states from CEA-1 — while less elegant for CEA-2 it makes modifications easier to describe).

### 2.1.3 Cube Orientations

For  $k > 1$ , let  $f_k$  be the face shared by  $P_{1\dots k-1}$  and cube  $c_k$ . (Recall that  $f_1$  was previously defined explicitly as the original exposed face.) For all  $k \geq 1$ , we call  $f_k$  the **bottom face** of  $c_k$  and the opposite face in  $c_k$  to be the **top face** of  $c_k$ .

The remaining faces of  $c_k$  are called *side faces*, and they follow the cycle **up face**, **right face**, **down face**, **left face** around the cube in a clockwise direction relative to the top face. Each cube has an *absolute orientation* defined by the normal vectors  $v_u$  and  $v_t$  of the up and top faces respectively ( $v_u, v_t \in D_{PO}$ ). The top face of  $c_1$  is the face opposite  $f_1$  as described earlier, and we call the corresponding vector  $v_t$  the *extrusion vector* as it intuitively describes the direction that the cube is extruded. We let the north-facing face of  $c_1$  be the up face — there is no obvious intuitive interpretation of this vector as it is just picked arbitrarily once in order to be consistent throughout. As an example, for the base case  $c_1$ ,  $v_t$  is above and  $v_u$  is north.

## 2.2 Paper Allocation

Here we determine the size of the required sheet, and which squares of the unfolded sheet correspond to faces on the polycube.

### 2.2.1 Face Grid

Define a *face grid* as a geometric graph with *vertices* embedded as the centers of unit squares on the two-dimensional square grid, and *edges* corresponding to shared edges between the squares. We require that a face grid be a complete  $m \times n$  rectangle. Squares in the face grid can be referred to by their indices e.g.  $s_{i,j}$ , but those indices are not stored and maintained so it takes  $O(i+j)$  time to find a square referenced by its coordinates. We do however store the grid edges which can be referred to by their direction: **up** (u), **down** (d), **left** (l) or **right** (r). This set of *square orthogonal directions* is denoted as  $D_{SO}$ ; they will end up corresponding to the side face names described above. The origin square  $s_{1,1}$  is the most down and left square, and the square in the opposite corner is  $s_{m,n}$ .

A face grid represents the unfolded sheet of paper and keeps track of the correspondences and relative orientations of squares in the unfolded sheet and faces in the polycube. For each step (extrusion) there is a corresponding face grid  $S_k$ . Formally,

a face grid  $S_k$  must satisfy the **face correspondence invariant** — there must be a map between all faces of  $P_{1\dots k}$  (except for  $f_1$ ) and squares in  $S_k$ . Also, one useful operation which can be applied to a face grid is the insertion of rows or columns. In other words, one can add rows or columns of squares, shifting the existing squares up or right respectively.

## 2.2.2 Generating the Face Grid Sequence

A particular folding of a polycube is completely described by a **sequence of face grids**  $S_1 \dots S_N$ . Now we inductively describe how to generate these face grids based on the desired polycube, the extrusion order, the absolute orientations of previously extruded cubes, and the previous face grid.

The base case is a one-cube polycube. The corresponding  $S_1$  has  $m$  and  $n$  being  $2r + 1$  and  $2c + 2$  respectively (where  $r$  and  $c$  are the parameters of the type of cube gadget to be used to form the cube). The top face of  $c_1$  corresponds to  $s_{r+1,c+1}$  in  $S_1$ . The up face of  $c_1$  corresponds to the adjacent square in the up direction of  $s_{r+1,c+1}$  in  $S_1$  ( $s_{r+2,c+1}$ ). The right face of  $c_1$  corresponds to the adjacent square right of  $s_{r+1,c+1}$  in  $S_1$  ( $s_{r+1,c+2}$ ). This is similarly true for the remaining two side faces. The bottom face of  $c_1$  ( $f_1$ ) is an “exception” — it is the only face of the polycube that does not have a corresponding face in the face grid sequence (as a final folding step,  $s_{r+1,2c+2}$  will cover it, but for now it is convenient to ignore it).

Next, given face grid  $S_k$  corresponding to polycube  $P_{1\dots k}$ , we would like to derive the face grid  $S_{k+1}$  corresponding to the polycube  $P_{1\dots k+1}$ . Figures 1-3 and 1-4 give some intuition for how works. Let  $S_{temp}$  be a duplicate of  $S_k$ . Face  $f_{k+1}$  already has a corresponding square  $\sigma$  in  $S_{temp}$  as it is a top or side face of an existing cube. In  $S_{temp}$  insert  $r$  rows up and down relative to  $\sigma$  and insert  $c$  columns to the left and right of  $\sigma$ . The face of  $c_{k+1}$  opposite  $f_{k+1}$  now corresponds to the square  $\sigma$  (as the face that  $\sigma$  previously corresponded to is now covered by the cube).

We derive the absolute orientation of  $c_{k+1}$  from the absolute orientation of its **parent cube**  $c_p$ , defined as the cube in  $P_{1\dots k+1}$  that shares the face  $f_{k+1}$  with the cube  $c_{k+1}$ . If  $f_{k+1}$  is the top face of  $c_p$  then the absolute orientation of  $c_p$  and  $c_{k+1}$

are the same (i.e., the up face and top face point in the same direction). Otherwise,  $f_{k+1}$  is a side face of  $c_p$ , and situation is more complicated.

In that case we take the vectors which describe the absolute orientation of  $c_p$ , copy them, and rotate them  $90^\circ$  along the appropriate axis such that the  $v_t$  vector is normal to the top face of  $c_{k+1}$ .<sup>2</sup> These new vectors describe the absolute orientation of  $c_{k+1}$  which determine which side faces are which. This information can then be used to create the correspondences as described in the base case (i.e., the up face corresponds to the square in the up direction of  $\sigma$  in the face grid).

We now have all the faces of  $P_{1\dots k+1}$  (except  $f_1$ ) in correspondence with the squares in  $S_{temp}$ , so we rename  $S_{temp}$  to  $S_{k+1}$ , adding it to our face grid sequence (as it now satisfies the face correspondence invariant). We also have absolute orientation for all the cubes  $c_1 \dots c_{k+1}$ . Having satisfied the inductive statement we can generate the full face grid sequence from 1 to  $N$ .

### 2.2.3 Cube Gadget Sequences

Now we generate a *sequence of symbolic cube gadgets*. A *symbolic cube gadget* has the form  $CubeGadget(type, (m, n), s_{i,j})$ . The idea is that each symbolic cube gadget injectively maps to a particular folded state with one cube gadget, without directly giving any details contained in the cube gadget diagram (that detail is added in the fold pattern). This is the last strictly combinatorial part of the algorithm.

Let  $G_k$  be the cube gadget corresponding to face grid  $S_k$ . **Type** refers to the type of cube gadget (which defines the folded state and  $[r, c]$ ),  $m$  and  $n$  come from the dimensions of  $S_k$ , and  $s_{i,j}$  is the square in  $S_k$  corresponding top face of  $c_k$ . As an example, the symbolic representation of a cube-gadget sequence for a polycube of two cubes could be  $\langle CubeGadget(slit, (3, 4), s_{2,2}), CubeGadget(slit, (5, 6), s_{3,3}) \rangle$ .

---

<sup>2</sup>These two cases and the  $90^\circ$  tilt are a result of folding “motion” which can be most clearly seen in Figure 1-3.

## 2.3 Folding Representation

Representing a three-dimensional folded state in a combinatorial way that can be manipulated and coalesced is complex, and we neither describe that here nor have implemented it (the standard model of paper does not represent the intricacies of such a structure). We expect to publish a paper after this thesis is complete exploring this issue in more depth, but for now we make do with a more vague but intuitive description. Note that, for practical purposes of both origami design and robotics, the folded state does not have to be represented as a combinatorial structure — we can also represent folding in a human-readable form (origami diagrams), or through a kinematic simulation. This is why we say that the combinatorial section ends with cube gadget sequences, which already completely describe the folding process when paired with a general cube-gadget diagram.

### 2.3.1 Fold Pattern Sequences

A *fold pattern* is an explicit (data structural) representation of a folded state. It is an angle pattern together with an overlap ordering for all overlapping *facets* (faces broken up when necessary to give a unique overlap ordering). A fold pattern is the unambiguous extension of the idea of a crease pattern. Like a crease pattern it describes the folded shape without explicitly specifying any coordinates, but unlike a crease pattern the angle information completely specifies the folded geometry and the overlap ordering information eliminates the remaining ambiguities.

The details of a fold pattern can be described by a *diagram* drawn on a piece of paper with *boundary arrows* (see Figure 1-2). The arrows at the boundary of the piece of paper shown can continue until they reach the sides of the paper and the shaded region delimits the paper used by the actual cube gadget. We refer to the intuitive process of redrawing a diagram with boundary conditions on a partial (possibly empty) piece of paper at a specified location as *stamping* a diagram.

The symbolic cube-gadget sequences generated above can be evaluated to give a *fold-pattern sequence* — a list of fold patterns  $F_1, F_2, \dots, F_N$  corresponding to the evaluation of cube gadgets  $G_1, G_2, \dots, G_N$ . Given a cube gadget  $G_k$  of the form

$CubeGadget(type, \{m, n\}, s_{i,j})$ ,  $F_k$  specifies how to fold an  $m \times n$  rectangle into an  $(m - 2c) \times (n - 2r + 1)$  rectangle with a cube on the square  $s_{i-c,j-r}$ .

### 2.3.2 Complete Fold Pattern

Here we describe how to generate a **complete fold pattern** which describes the folding of the entire polycube from a rectangular starting sheet (via a composition of fold patterns from the fold-pattern sequence). First, we give an intuitive description of what humans would do to actually fold the polycubes — this gives the flavor for what should be done computationally, though it glosses over many details.

We start at the last step  $F_N$  of the fold-pattern sequence and fold a piece of paper of the size described according to the fold pattern. After folding a cube gadget at the specified location, the paper will have the same dimensions as required by  $F_{N-1}$  except it will have a folded cube on it and pleats radiating from the cube. These modifications to the rectangle do not matter for subsequent folding — pleats are folded flat so they can be folded through, and cubes are never folded through as they are made of faces (which must remain seamless by the argument in Lemma 1). Now we fold following the fold pattern  $F_{N-1}$ . This yields a smaller rectangle with more cubes and pleats. We continue going backwards through fold patterns until we complete the final fold pattern  $F_1$  (as described earlier, the paper may temporarily self intersect — in order to avoid this, one can simply “squash” cubes down temporarily if they would intersect and “unsquash” them when it is safe to do so). The final step is to fold back the extra flap to seamlessly cover  $f_1$ .

The computational description of generating the complete fold pattern is analogous. Starting with the fold pattern for  $F_N$ , we iteratively apply the subsequent fold patterns  $(F_{N-1}, \dots, F_1)$  to generate the complete fold pattern. We treat the intermediate states of the complete fold pattern as “**bumpy paper**,” meaning that they can be folded like a simple rectangle of paper. Even though intermediate states will have bumps (cubes and pleats), we know it is safe to ignore cubes and treat pleats as a single layer when simulating folding. As described above, the final step is to fold back the extra flap (originally  $s_{r+1,2c+2}$ ) to seamlessly cover  $f_1$ .

## 2.4 Performance

As was true of the CEA-1 (Section 1.3), this more detailed implementation folds an  $N$ -cube polycube from a  $(2rN + 1) \times (2cN + 2)$  rectangle of paper by a sequence of  $N$  applications of an  $[r, c]$ -cube gadget. However, with this more explicit description of the basic algorithm we can give a more nuanced interpretation of the run-time of the *combinatorial section*, which is polynomial, or more specifically  $O(N^2)$ .

The first step is finding a spanning tree of the polycube graph, which takes  $O(N)$  time. Next, we generate the face grid sequence, which involves adding a constant number of rows and columns to a two-dimensional linked list for each of  $O(N)$  steps. This takes  $O(N^2)$  time in total (there is also some constant-time computation to determine absolute orientations). Determining the parameters for each cube gadget in a sequence takes  $O(N)$  time. This gives us a total time of  $O(N^2)$  to generate a complete sequence for folding a polycube.

We do not describe in detail how the actual fold pattern is constructed so we cannot give a worst-case bound, but we can conjecture what a lower bound might be based on some observations about the structure of our fold patterns. We note that there can be at most  $O(mn)$  angles and  $O(mn)$  facets in a fold pattern of size  $m \times n$  even if we use all the hinges in a hinge pattern. So if we represented each hinge separately and used a linked list to store the stacking order of all facets, the most complex folding would use only  $O(mn)$  components. In particular, the fold patterns for a single cube gadget use as few as  $\Theta(m+n)$  components ( $\approx 2r$  horizontal and  $\approx 2c$  vertical with some constant number at their intersection). To generate the complete fold pattern, we iterate through all the folds of each fold pattern in a sequence, sometimes reflecting folds through multiple layers. In total this could take as little as  $\Theta(N(m+n))$  time.<sup>3</sup> Because  $m$  and  $n$  are both  $\Theta(N)$ , the total run-time could remain  $\Theta(N^2)$ .

---

<sup>3</sup>There are some subtleties here as a single fold can be reflected through  $O(N^2)$  layers, but we believe that might be dealt with via clever amortized analysis.



# Chapter 3

## Extensions

We now extend to folding *polygroves*, which are one or more polycubes resting at integral points on a rectangle in the  $xy$ -plane (see Figure 3-1). Polygroves are actually intermediary steps in folding a polycube, but being able to explicitly fold arbitrary polygroves is not possible with just CEA-2. In addition, modifications to CEA-2 which yield tighter foldings can be more easily described when applied to polygroves. Much of this chapter is dedicated to describing these improvements (see Figure 3-2 for an outline of the improvements).

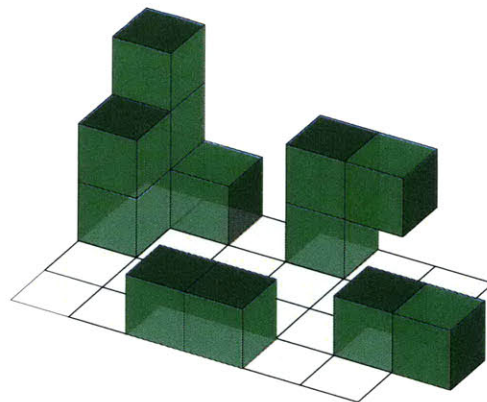


Figure 3-1: An example of a polygrove — one or more polycubes resting at integral points on a rectangle in the  $xy$  plane.

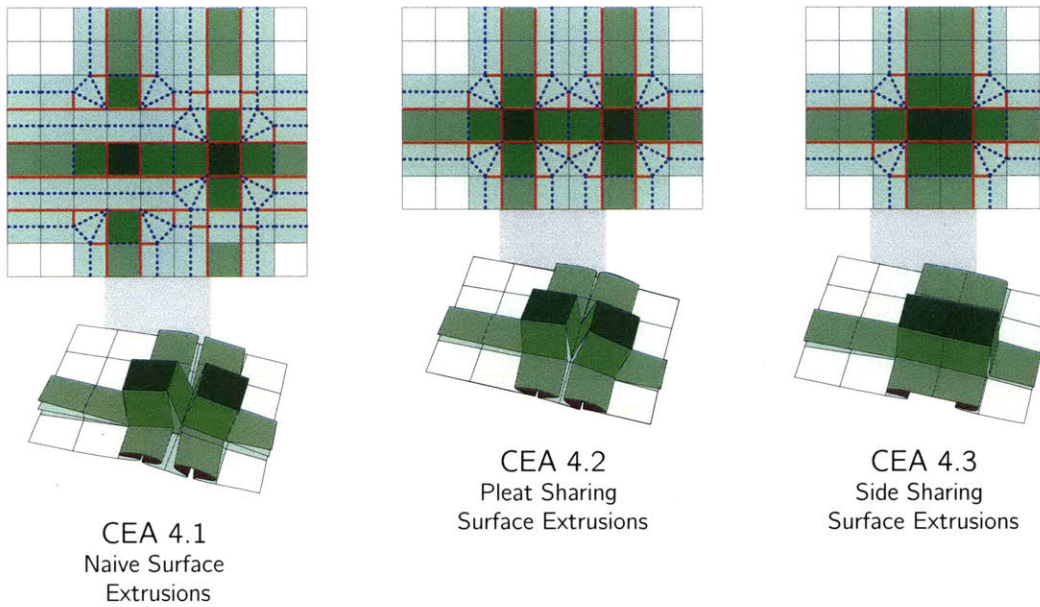


Figure 3-2: An example of how the incremental improvements to CEA-2 affect the mountain-valley pattern, rectangle size, and folded state of a simple two cube polygrove.

### 3.1 Naïve Surface Extrusions

*CEA-3.1* is an extension of the previous algorithm (CEA-2) that naïvely folds a polygrove — formally, a set of *subpolycubes* on an  $m^* \times n^*$  *base rectangle*. The subpolycubes each have root cubes with faces that touch (and connect) to specific squares of the base rectangle.

#### 3.1.1 Polygrove Representation

We represent the polygrove as a single *total polycube*  $P$ , with an  $m^* \times n^*$  rectangle of *dummy cubes* at coordinates in the grid between  $(0, 0, -1)$  and  $(m^* - 1, n^* - 1, -1)$ . The bottom face of each subpolycube is connected to the top face of a dummy cube and touching dummy cubes are connected to each other. We take a spanning tree  $T$  for the total polycube by treating the base rectangle as a single root (which connects to the root cubes of the subpolycubes). As before (Section 2.1.2), the extrusion sequence is the result of pre-order traversal of this spanning tree.

### 3.1.2 Face Grid Base Case

We generate face grids as before (Section 2.2.2), except our base case is an  $m^* \times n^*$  rectangle instead of an initial cube. We define a correspondence between the top face of each dummy cube to a square in the face grid, such that the top face of the cube at  $(i - 1, j - 1, -1)$  corresponds to square  $s_{i,j}$  in the face grid. Then we generate the face grid sequence (Section 2.2.2), cube gadget sequence (Section 2.2.3) and fold patterns (Section 2.3) as before.

## 3.2 Pleat-Sharing Surface Extrusions

We can do much better than this naïve surface extrusion algorithm (CEA-3.1) in specific common cases. For example, consider a  $4 \times 5$  base rectangle with a cube on  $s_{3,3}$  and on  $s_{4,3}$ . The naïve algorithm extrudes each cube separately, using  $2r$  rows and  $2c$  columns for each cube and thus  $4r$  rows and  $4c$  columns in total. However, by extruding the cubes simultaneously we can have the cubes *share pleats*, requiring only  $2r$  rows and  $4c$  columns (see Figure 3-3). Intuitively, this works because the cube gadget's only effect on the rest of the paper is to make half-square pleats radiating from the cube. If two cubes radiate pleats into the same row (or column) then they can use each other's pleats instead of forcing the creation of new pleats. We now describe a *pleat-sharing modification* of CEA-3.1 to get this more efficient algorithm which we call **CEA-3.2**.

### 3.2.1 Extrusion Strategy

The key to attaining this increased efficiency is to parallelize — to do simultaneous extrusions of cubes whose bottom faces share rows or columns in the face grid. Thus, instead of an extrusion sequence, we use an *extrusion set sequence*, which is a sequence of sets of cubes. We call a set of simultaneously extruded cubes an *extrusion set* (whether or not they share paper). Each extrusion set corresponds to a step, and the  $k$ th extrusion set is denoted  $E_k$  with cubes  $\{c_k^1, c_k^2, \dots\}$ . We let

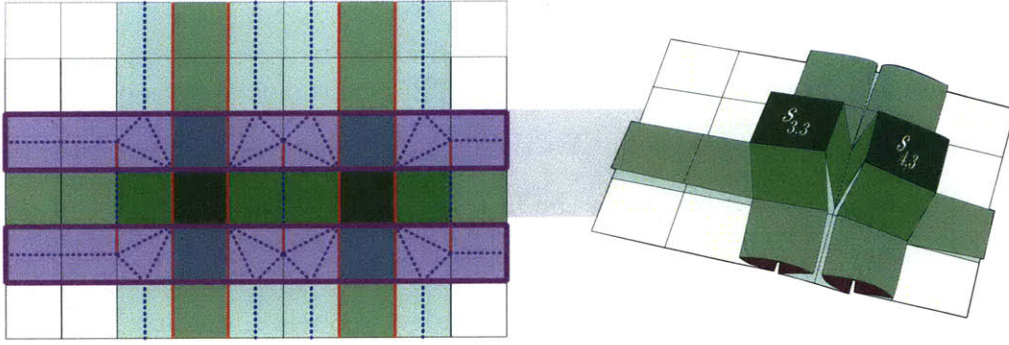


Figure 3-3: The mountain-valley pattern and folded form associated with the pleat-sharing example described in Section 3.2. Note that the highlighted rows are shared by two cubes.

$P_{1\dots k}$  be the subpolygrove containing the base rectangle and extrusion sets  $E_1 \dots E_k$  (so  $P_{1\dots k}$  is a valid (connected) subpolygrove).

We call the algorithm for determining the step in which we extrude each cube the **extrusion strategy**. An extrusion strategy must produce an extrusion set sequence such that it only extrudes cubes from a face of the already-built polycube. More formally, each of the cubes in  $E_{k+1}$  must be connected via a face in  $P_{1\dots k+1}$  with at least one cube in  $P_{1\dots k}$ . The spanning tree used in Section 2.1.2 had this property, and a level-by-level breadth-first search is the parallel analogue. The **breadth-first extrusion strategy** does a simple breadth-first search of  $T$ , adding each level of the search as a new extrusion set. It is greedy, so it may not always produce the smallest fold pattern, but it is simple and fast.

### 3.2.2 Pleat Allocation

After picking and executing our extrusion strategy, we generate face grids. The base case is the same as above ( $m^* \times n^*$  rectangle), but the process of generating nontrivial grids is more complicated. Given the face grid  $S_k$  for step  $k$ , we need to determine how many rows and columns to insert to obtain the face grid  $S_{k+1}$  for step  $k+1$  (and where those rows and columns should be inserted). The idea is to keep track of which pleats are needed, each of which corresponds to a row or column insertion, and keep only unique pleats (as we remove duplicate pleats we instead have multiple

cubes share the same pleats).

A **pleat allocation**  $A_{od}^a$  has an orientation  $o$ , coordinate  $a$ , and depth  $d$ . Intuitively, a pleat allocation describes where a pleat will need to be. We say a pleat **points** in a direction in  $D_{SO}$  if that is the direction from the mountain crease of the pleat to the valley crease of the pleat. The **orientation** describes whether the pleat will point up, down, left, or right (in the face grid). The **coordinate** refers to the row or column index that the pleat is inserted relative to (in the face grid of step  $k$ ). Finally, the **depth** refers to how many pleats added in this step are “above” that pleat. For example,  $A_{r_1}^3$  denotes a pleat insertion directly (at depth 1) “right” of the third row, and  $A_{r_2}^3$  denotes pleat insertion to the right of that pleat (depth 2; under  $A_{r_1}^3$  in the folded state). Each step has a pleat-allocation set  $\phi_k$  to keep track of which pleats were already inserted in that step.

As before (Section 2.2.2), given a face grid  $S_k$  corresponding to polygrove  $P_{1\dots k}$ , we would like to derive the face grid  $S_{k+1}$  corresponding to the polygrove  $P_{1\dots k+1}$ . Also as before we let  $S_{temp}$  be a duplicate of  $S_k$  and manipulate it until it satisfies the face correspondence invariant. We iterate over cubes  $c_k^1, c_k^2, \dots$  in the extrusion set  $E_k$ . Each cube  $c_k^t$  has a bottom face  $f_k^t$  with corresponding square  $\sigma^t$  in  $S_{temp}$ . For each bottom face we allocate  $r$  pleats up and down relative to it and  $c$  pleats left and right. In particular, for  $\sigma^t$  at  $s_{i,j}$  in  $S_k$  and  $\rho$  rows from 1 to  $r$ , we add “up” pleats  $A_{u1}^i, A_{u2}^i, \dots, A_{u\rho}^i$  into  $\phi_{k+1}$  (and similarly for down, left, and right; except for columns we use the  $j$  index instead of  $i$ ). Because  $\phi_{k+1}$  is a set, replicated pleats are avoided.

### 3.2.3 Generating Face Grids

We now make a map from indices  $1 \dots m$  to the first square in each row of  $S_{temp}$  (and do the corresponding operation for columns). This map ignores changes in the locations of squares, so we use it to insert all the allocated pleats in the appropriate places (even though the insertions cause the actual indices of the squares to change). Next, we map each polygrove square to a face grid face as before (Section 2.2.2). This causes the face correspondence invariant to be satisfied, so we rename  $S_{temp}$  to  $S_{k+1}$  and have completed the induction.

### 3.2.4 Generating Cube-Gadget Steps

Multiple cube gadgets used in a single step are represented by single symbolic statement  $G_k^* = \text{CubeGadgetStep}(\text{type}, (m, n), s_{i_1, j_1}, s_{i_1, j_2}, \dots)$  with all the  $\sigma_t$ 's present where previously there was only one. For example,  $\langle \text{CubeGadgetStep}(\text{slit}, (3, 7), s_{2,2}, s_{2,6}), \text{CubeGadgetStep}(\text{slit}, (5, 11), s_{3,1}, s_{3,2}, s_{3,3}) \rangle$ .

### 3.2.5 Generating Pleat-Sharing Fold Pattern

The fold pattern is described in the same way as before (Section 2.3), except in this case we stamp down multiple cube gadget diagrams simultaneously onto the same piece of paper. We say that the boundary arrows of two stamped diagrams **match** if arrows of separately stamped diagrams only intersect head on if they have the same angle. By inspection, the cube gadget diagrams can tile a plane of  $2r+1 \times 2c+1$  rectangles, or any subset of it, such that the boundary arrows all match up. So we stamp *type* cube gadgets on an  $m \times n$  sheet of paper with cube gadgets centered at  $s_{i_1, j_1}, s_{i_1, j_2}, \dots$  to create the fold pattern for  $\text{CubeGadgetStep}(\text{type}, (m, n), s_{i_1, j_1}, s_{i_1, j_2}, \dots)$ .

## 3.3 Side-Sharing Surface Extrusions

In addition to sharing pleats, it is also possible to **share sides**. For example, consider again the  $5 \times 4$  base rectangle with a  $2 \times 1 \times 1$  polycube connected to the base rectangle at  $s_{3,3}$  and  $s_{4,5}$  (see Figure 3-4). The pleat-sharing modification (with a breadth-first extrusion strategy) extrudes the two cubes of the polycube in a single step, sharing their row pleats. However, even though the two cubes are connected by their side faces in the polycube data structure, previous algorithms create unnecessary gaps between cubes that use up two column pleats of paper. **CEA-3.3** eliminates some of that waste by sharing sides of cubes to form raised rectangles. Note that this also changes the structure of the folded polycube to be more “connected” than was possible by the previous algorithms, where each cube in the folding was only physically connected to

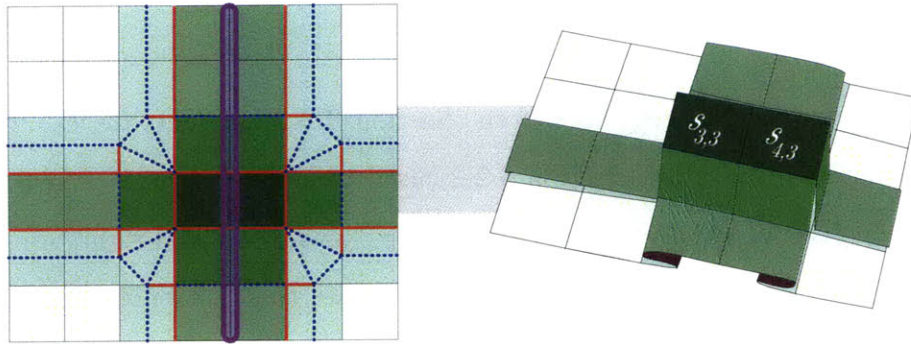


Figure 3-4: The mountain-valley pattern and folded form associated with the side-sharing example described in Section 3.3. Note that the highlighted area denotes where columns are removed due to sides shared by the two cubes.

a single parent.<sup>1</sup>

### 3.3.1 Cube Gadget Wings

We first break down a cube gadget into its component parts so that we can describe the side-sharing process more clearly. Each of our cube gadgets has a **center square**, which is the top face in the folded form, and one **wing** in each  $D_{SO}$  direction, which is the paper used to make the side faces of the cube (the actual side face in direction  $\delta$  is the unit square directly  $\delta$  of the center square). Beyond that the boundary arrows can be extended indefinitely until they hit the end of the paper or other boundary arrows of the same angle going in the opposite direction.

A cube gadget can be modified to remove its wings and is described by a **wing set**  $\omega$ , where  $\omega$  is just some subset of  $D_{SO}$  delimiting which wings are not present. More formally, a cube gadget with a wing set  $\omega$  has one region removed for each element  $\delta$  in the wing set. This is the halfplane extending in the  $\delta$  direction from the line containing the edge of the center square in the  $\delta$  direction. All angles on the border are also removed and the resulting segment endpoints become boundary arrows. These **sub-cube gadgets** described by their wing sets are used to avoid needing to allocate pleats for the shared sides of cubes.

<sup>1</sup>Note that in order to maximize “connectedness” a more sophisticated extrusion strategy is needed.

### 3.3.2 Side-Sharing Strategy

We use an extrusion strategy as before (Section 3.2.1) to generate  $E_1, \dots, E_k$ , but the pleat-allocation process (Section 3.2.2) used to generate the face grid is slightly modified. Instead of simply inserting pleats up, down, left, and right, we first check to see if we can share some sides (with other cubes that are being extruded at the same time), and if so, we do not insert pleats in the corresponding directions.

A **connected minimal square** of a polycube is exactly four cubes such that they form the smallest simple cycle possible for a polycube graph (as described in Section 2.1). Formally, a cube  $c_{k+1}^{t1}$  in  $E_{k+1}$  and a direction  $\delta$  in  $D_{SO}$  satisfies the **local side-sharing condition** if there exists another cube  $c_{k+1}^{t2}$  in  $E_{k+1}$  such that their parent cubes have the same extrusion vectors and all four cubes form a connected minimal square (note that the dummy cubes are necessary for this condition).

Cube gadgets with convex corners only allow us to extrude rectangles of shared sides, so it can be the case that a side that can be shared locally is not able to be shared given other sides that are shared, if that would extrude a non-rectangle with shared sides in a single step. The **side-sharing strategy** determines which sides are actually shared, from among those which satisfy the local side-sharing condition. For example, the **down-right side-sharing strategy** builds rectangles greedily by starting at the uppermost leftmost face-grid square to be extruded, and marking rectangles as large as possible by first going down as far as possible and then right as far as possible, ignoring rectangles it has already marked. This strategy is greedy, and may not always produce the smallest fold pattern, but is simple and fast.

The side-sharing strategy annotates edges of the face grid with a **share mark** to denote that the sides of the cubes extruded from those squares should be shared. For example, if the cube  $c_k^{t1}$  (at  $s_{i,j}$ ) could share its right side with the left side of cube  $c_k^{t2}$  (at  $s_{i,j+1}$ ), with a [1,1]-cube gadget, then the pleats  $A_{l,j}^a$  and  $A_{r,j+1}^a$  would not be needed, so the edge between the corresponding face grid squares  $f_k^{t1}$  and  $f_k^{t2}$  would receive a share mark.



### 3.3.3 Generating Side-Sharing Cube-Gadget Steps

We create a symbolic representation of the folded form as in 3.2.4, except that we encode information about side sharing. In particular, we modify our symbolic representation from Section 3.2.4 to have a tuple  $(s_{i,j}, \omega)$  in place of  $s_{i,j}$ , where  $\omega$  is the wing set described earlier — intuitively, here it is just some subset of  $D_{SO}$  delimiting the directions that a cube gadget is shared. This information is easily extracted from share marks. For example,  $CubeGadgetStep(\arctan \frac{1}{2}, (5, 4), (s_{3,3}, \{r\}), (s_{4,3}, \{l\}))$ .

### 3.3.4 Generating Side-Sharing Fold Pattern

Because we do not allocate enough paper to do so, we can obviously not stamp entire cube gadgets as we did previously. Instead, we center cube gadgets as before, but only stamp the unshared sides (as denoted by the  $CubeGadgetStep$ ). More formally, we only stamp the sub-cube gadget specified by the given wing set  $\omega$ , with the center square at the given location  $s_{i,j}$ . Boundary arrows match, because either sides are not shared, which is the same as before, or sides are shared, in which case the corresponding wings are removed, so that the edges of center squares touch (which also matches by inspection).

## 3.4 Performance

We now examine “paper size performance” of CEA-3.1 through CEA-3.3, and also the run-time performance for the combinatorial parts of the algorithms.

### 3.4.1 Paper Size

Here we discuss optimality of paper size informally as vague conjectures that might provide insight into future research. This thesis is focused on constructions, not detailed analysis, but we feel it would be helpful to share our intuitions. When we conjecture that an algorithm produces optimal folded states, we mean that the size

of the rectangular paper used by the folded states is the smallest possible paper size to fold that polycube given any possible hinge pattern (without slits).

CEA-3.1 folds an  $N$ -cube polygrove from a  $(2rN + m^*) \times (2cN + n^*)$  rectangle of paper by a sequence of  $N$  applications of an  $[r, c]$ -cube gadget, and is optimal in the same respects as CEA-1 (Section 1.3) and CEA-2 (Section 2.4). In addition, we conjecture that CEA-3.1 produces optimal folded states when the cubes to be extruded are sufficiently sparse and “never aligned.”

CEA-3.2 is less wasteful and uses only a  $(2rN + m^* - r^\#) \times (2cN + n^* - c^\#)$  rectangle of paper for the same folding, where  $r^\#$  and  $c^\#$  are the number of removed pleats, and are bounded by the total number of pleats that could be used. In a scenario where many pleats are shared, such as when extruding a  $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$  grid of equally spaced disconnected cubes, we can share  $\sqrt{N}$  pleats on every row and column (for integral  $\sqrt{N}$ ). This makes  $r^\# = 2r(N - \sqrt{N})$  and  $c^\# = 2c(N - \sqrt{N})$ , giving a rectangle of size  $(2r\sqrt{N} + m^*) \times (2c\sqrt{N} + n^*)$ , which ignoring  $m^*$  and  $n^*$  is a  $N \rightarrow \sqrt{N}$  savings. We conjecture that CEA-3.2 is optimal when CEA-3.1 is optimal, and also in many cases when the cubes of the polycube do not have connected sides or are “slightly misaligned.”

CEA-3.3 is even less wasteful and uses only a  $(2rN + m^* - r^\$ - r^\$) \times (2cN + n^* - c^\$ - c^\$)$  rectangle of paper for the same folding, where  $r^\$$  and  $c^\$$  are the number of rows and columns of removed sides (wings), and are bounded by the total number of wings that could be used. In a scenario where many sides are shared, such as when extruding a  $\sqrt[3]{N} \times \sqrt[3]{N}$  “supercube” of fully connected cubes, the generated folded state requires only a  $2r\sqrt[3]{N} + m^* \times 2c\sqrt[3]{N} + n^*$  rectangle of paper, which ignoring  $m^*$  and  $n^*$  is a  $N \rightarrow \sqrt[3]{N}$  savings. We conjecture that CEA-3.3 is optimal when CEA-3.2 is optimal, and also now in some cases where the cubes of the polycube have connected sides, especially if there are no concave angles in the desired polycube.

### 3.4.2 Run-time

The main difference between CEA-2 and CEA-3.1 is that the  $m^* \times n^*$  grid needs to be taken into account. A simplistic direct implementation of CEA-3.1 would therefore

increase in run-time by  $O(m^* + n^*)$  for each step, which would yield a total run-time of  $O(N(N + m^*n^*))$ . This could potentially be improved with a slightly more clever “sparse” data structure.

CEA-3.2 just needs additional time to determine the extrusion strategy (everything else including pleat allocation is with respect to the number of cubes or steps as before). A simple breadth-first search extrusion strategy takes only  $O(N)$  time, so with such a strategy run-time remains  $O(N(N + m^*n^*))$  (in fact, it is likely to “feel” faster than CEA-3.1 as inserting less pleats saves time).

Finally, CEA-3.3 just needs additional time for the side-sharing strategy. The down-right side-sharing strategy described looks only once at each square of a rectangle of cubes to extrude, and may also look at one square outside a rectangle in order to determine whether it has gone too far. This takes only a constant amount more time than the total number of cubes to be extruded each step, so it takes  $O(N)$  time total, still keeping the total run-time  $O(N(N + m^*n^*))$ .



# Chapter 4

## Implementation

Here we describe our algorithmic implementations, and how we use those implementations with other tools to partially automate the folding process. Figure 4-1 overviews the whole process.

### 4.1 Ruby Implementation

The combinatorial elements of all the algorithms described earlier were implemented in Ruby exactly following the descriptions. A programmer can manually describe the complete polygrove structure by specifying which cubes exist in the cube grid graph, and where there are edges between touching cubes. Alternatively, a non-programming user can extrude cubes one extrusion step at a time using a GUI. Either way the implementation will share pleats and sides when possible within a

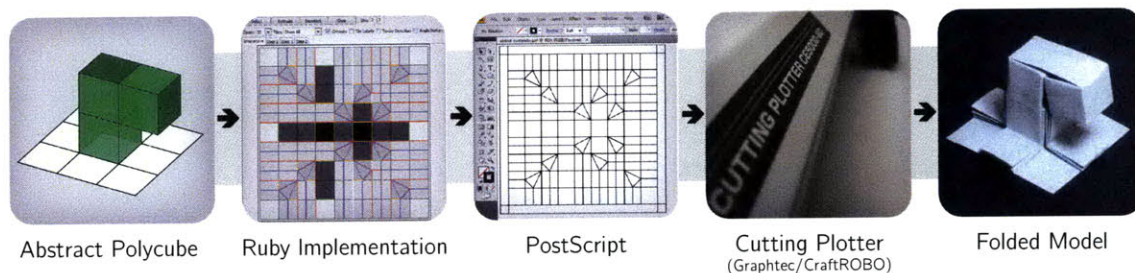


Figure 4-1: The process by which paper origami can be constructed using the described algorithms.

step (using a simple side-sharing strategy).

The output is either a folding sequence that can be followed as described in Section 2.3.1, or a *simplified composition of the folding sequence* which can be saved to a PostScript file. The simplified composition just shows the original angle pattern for each square tile on the full sheet of paper (i.e., the angle pattern from the step when the square was inserted as part of a row or column), and does not take into account non-orthogonal creases made as a result of later steps.

## 4.2 Partial Folding Automation

The PostScript file can be sent to a cutting plotter, which can etch the simplified composition onto a sheet of paper. Alternatively, one can use a laser cutter to score the paper. Once etched or scored, the step-by-step sequence generated by the implementation can be “simply” followed until the final folded form is reached, though it may require a lot of skill to make all the simultaneous and multilayer folds required.

Artistic origami produced by this method has been displayed at various convention galleries, student galleries, and even an airport gallery (Figure 4-2).

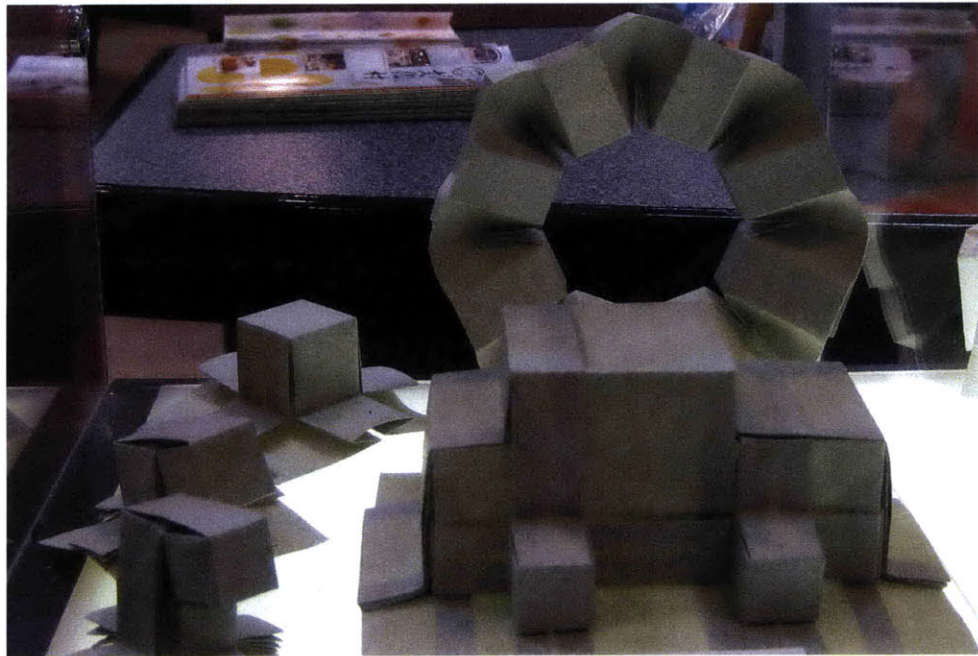


Figure 4-2: A display at the Origami Museum at Narita Airport, Tokyo. Left: “Cube Evolution,” a succession of three simple polycubes from back to front, each with one additional cube (requires CEA-3.1). Back: “Inverse Cube Ring” which was designed with a single extrusion step with shared pleats and turned inside out (requires CEA-3.2). Right: “Low Resolution Car” which was designed with three extrusion steps with shared sides and pleats (requires CEA-3.3).





# Bibliography

- [1] Marshall Bern and Barry Hayes. The complexity of flat origami. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 175–183, Atlanta, January 1996. 13
- [2] Erik D. Demaine, Martin L. Demaine, and Joseph S. B. Mitchell. Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami. *Computational Geometry: Theory and Applications*, 16(1):3–21, 2000. 14
- [3] Erik D. Demaine, Satyan L. Devadoss, Joseph S. B. Mitchell, and Joseph O’Rourke. Continuous foldability of polygonal paper. In *Proceedings of the 16th Canadian Conference on Computational Geometry*, pages 64–67, Montréal, August 2004. 21
- [4] Erik D. Demaine and Joseph O’Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, July 2007. 19, 20
- [5] Erik D. Demaine and Tomohiro Tachi. Origamizer: A practical algorithm for folding any polyhedron. Manuscript, 2010. 14
- [6] E. Hawkes, B. An, N. M. Benbernou, H. Tanaka, S. Kim, E. D. Demaine, D. Rus, and R. J. Wood. Programmable matter by folding. *Proceedings of the National Academy of Sciences of the United States of America*, 107(28):12441–12445, 2010. 9, 14
- [7] Robert J. Lang. *Origami Design Secrets: Mathematical Methods for an Ancient Art*. A K Peters, 2003. 13
- [8] Robert J. Lang. Computational origami: from flapping birds to space telescopes. In *Proceedings of the 25th Annual Symposium on Computational Geometry*, pages 159–162, Aarhus, Denmark, 2009. 13



# Index

- $A_{od}^a$ , 45
- $C_k$ , 21
- $D_{PO}$ , 34
- $D_{SO}$ , 35
- $E_k$ , 43
- $F_N$ , 39
- $F_k$ , 20, 39
- $G_k^*$ , 46
- $G_k$ , 37
- $N$ , 27
- $P_{1\dots k}$ , 34
- $P$ , 21
- $S_k$ , 35
- $T$ , 28
- $\omega$ , 47
- $\phi_k$ , 45
- $\sigma^t$ , 45
- $\sigma$ , 24
- $c_1$ , 34
- $c_k^t$ , 45
- $c_k$ , 34
- $c_p$ , 36
- $c$ , 22
- $f_1$ , 34
- $f_k^t$ , 45
- $f_k$ , 34
- $i$ , 19
- $j$ , 19
- $m$ , 19
- $n$ , 19
- $r$ , 22
- $s_{i,j}$ , 19
- $t_{i,j}$ , 29
- $v_t$ , 35
- $v_u$ , 35
- above, 34
- absolute orientation, 35
- angle pattern, 20
- $\arctan \frac{1}{2}$  gadget, 23
- $\arctan \frac{1}{2}$  pattern, 23
- base rectangle, 42
- below, 34
- bottom face, 34
- boundary arrows, 38
- box-pleated gadget, 23
- box-pleated pattern, 20
- breadth-first extrusion strategy, 44
- bumpy paper, 39
- CEA-1, 19, 27

CEA-2, **33**  
 CEA-3.1, **42**  
 CEA-3.2, **43**  
 CEA-3.3, **46**  
 center square, **47**  
 coalesce folded state, **21**  
 coalesce result, **21**  
 coalesce sequence, **21**  
 coalesce set, **21**  
 complete fold pattern, **39**  
 connected minimal square, **48**  
 crease, **20**  
 crease pattern, **20**  
 cube gadget, **22**  
  
 diagram, **38**  
 down, **35**  
 down face, **35**  
 down-right side-sharing strategy, **48**  
 dual graph, **21**  
 dummy cubes, **42**  
  
 east, **34**  
 edges  
     face grid, **35**  
     polycube, **33**  
 exposed, **33**  
 extrusion sequence, **34**  
 extrusion set, **43**  
 extrusion set sequence, **43**  
 extrusion strategy, **44**  
  
 extrusion vector, **35**  
  
 face correspondence invariant, **36**  
 face grid, **35**  
     sequence, **36**  
 faces, **21**  
 facet, **38**  
 final folded state, **20**  
 final folded state of a coalesce sequence,  
     **21**  
 fold pattern, **38**  
 fold-pattern sequence, **38**  
 folded geometry, **20**  
 folded state, **20**  
 folding of a polycube, **21**  
 folding sequence, **20**  
  
 generated, **29**  
  
 half-square pleats, **22**  
 hinge, **19**  
     trivial, **20**  
 hinge pattern, **19**  
  
 inner tiles, **29**  
  
 left, **35**  
 left face, **35**  
 local side-sharing condition, **48**  
 location, **34**  
  
 match, **46**  
  
 north, **34**

null edge, **33**  
 number of layers at a point, **20**  
 number of layers of a folded state, **21**  
 parent cube, **36**  
 piece of paper, **19**  
 pleat  
     coordinate, **45**  
     depth, **45**  
     orientation, **45**  
     points, **45**  
 pleat allocation, **45**  
 pleat tiles, **29**  
 pleat-sharing modification, **43**  
 polycube, **21**  
     data structure, **33**  
 polycube orthogonal directions, **34**  
 polygrove, **41**  
 right, **35**  
 right face, **35**  
 root cube, **34**  
 seamless, **22**  
 share mark, **48**  
 share pleats, **43**  
 share sides, **46**  
 side faces, **35**  
 side-sharing strategy, **48**  
 simplified composition of the folding se-  
     quence, **54**  
 slit gadget, **23**  
 slit pattern, **23**  
 south, **34**  
 square orthogonal directions, **35**  
 stamping, **38**  
 starting sheet, **20**  
 step, **34**  
 step number, **34**  
 sub-cube gadgets, **47**  
 subpolycubes, **42**  
 symbolic cube gadget, **37**  
     sequence, **37**  
     type, **37**  
 tetrakis tiling, **20**  
 tile, **29**  
 tile set, **29**  
 top face, **34**  
 total polycube, **42**  
 transpose of a cube gadget, **31**  
 up, **35**  
 up face, **35**  
 vertices  
     face grid, **35**  
     polycube, **33**  
 west, **34**  
 wing, **47**  
 wing set, **47**