2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

# An End-to-End Approach to Making Self-Folded 3D Surface Shapes by Uniform Heating

Byoungkwon An*, Shuhei Miyashita*, Michael T. Tolley**, Daniel M. Aukes**, Laura Meeker*, Erik D. Demaine*, Martin L. Demaine*, Robert J. Wood** and Daniela Rus*

***Abstract*— This paper presents an end-to-end approach for creating 3D shapes by self-folding planar sheets activated by uniform heating. These shapes can be used as the mechanical bodies of robots. The input to this process is a 3D geometry (e.g. an OBJ file). The output is a physical object with the specified geometry. We describe an algorithm pipeline that (1) identifies the overall geometry of the input, (2) computes a crease pattern that causes the sheet to self-fold into the desired 3D geometry when activated by uniform heating, (3) automatically generates the design of a 2D sheet with the desired pattern and (4) automatically generates the design files required to fabricate the 2D structure. We demonstrate these algorithms by applying them to complex 3D shapes. We demonstrate the fabrication of a self-folding object with over 50 faces from automatically generated design files.**

## I. INTRODUCTION

In this paper, we develop an approach for the automatic creation of self-folded objects given a 3D geometric specification using print-and-fold processes. We have previously demonstrated the ability to accurately control the fold angle during uniform heating [1], [2]. In prior work, we considered defined shape memory laminate geometries capable of achieving target fold angles and demonstrated the self-folding of cylinders and regular polyhedra. In this paper, we generalize these results by showing that we can automatically generate the crease patterns and manufacturing files necessary to self-fold an arbitrary 3D geometry.

We examine this problem in two steps. First, we develop a suite of algorithms that start with the desired 3D geometry and automatically generate (1) the geometry of its corresponding 2D sheet, (2) the crease structure required to realize the 3D folded shape from the 2D sheet, (3) the mechanical design of a heat-activated self-folding device using the previously described edge folding angle control strategy [1], [2]. In the second step, we automatically generate the fabrication files required to produce and fabricate the device.

Just as an *origami crease pattern* contains the information required to produce a folded origami object, a *self-folding sheet design* contains information for automatically fabricating an object when subjected to uniform heating. We define these designs as a set of machine codes and develop a design algorithm for compiling an input 3D surface structure into its correlated mechanical design (Fig. 2). The design is composed of the graphic image of each layer. By printing (or cutting) and composing the layers of the output design, we build a self-folding sheet with embedded control program. Fig. 1 shows a self-folding bunny made from a 3D computer graphic model.

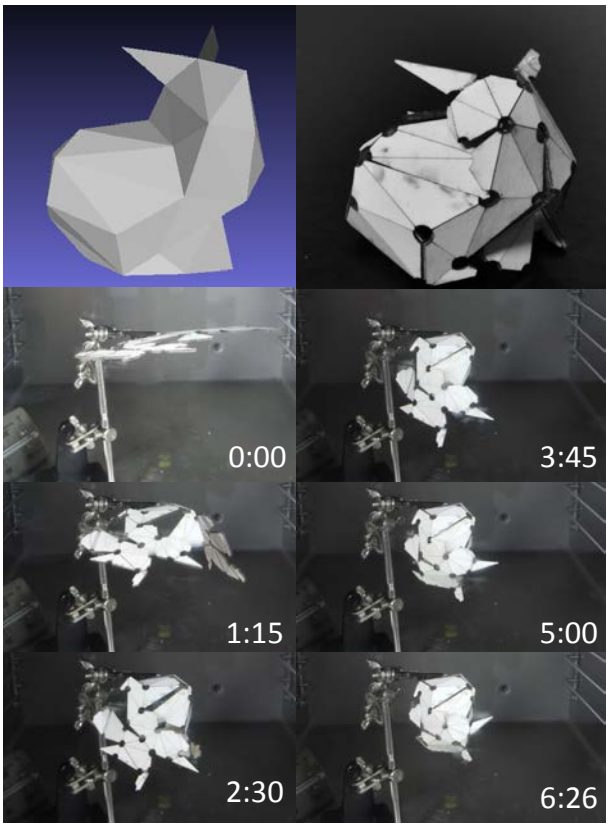


Fig. 1. Self-folding Stanford Bunny. (Top-left) Input 3D graphic model. (Top-right) 3D self-folded structure. (Bottom) Frames from experiment of self-folding by uniform heating. The time elapsed since exposure to uniform heating is indicated in the lower-right corner of each frame (in minutes and seconds).

We describe and analyze the self-folding models and the design algorithm in Sec. II, III. We explore the implementation of the algorithm and the actuation model in Sec. IV. Finally we demonstrate the experiments with four self-folding 3D structures in Sec. V. We discuss the conclusion and future works in Sec. VI.

Support for this work was provided in part by NSF grants EFRI-1240383 and CCF-1138967. We are grateful for it. We thank John Romanishin for insightful discussions on this research

* B. An, S. Miyashita, L. Meeker, E. D. Demaine, M. L. Demaine and D. Rus are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA dran@csail.mit.edu

** M. T. Tolley, D. M. Aukes and R. J. Wood are with the School of Engineering and Applied Sciences and the Wyss Institute for Biologically Inspired Engineering, Harvard University, Cambridge, MA 02138

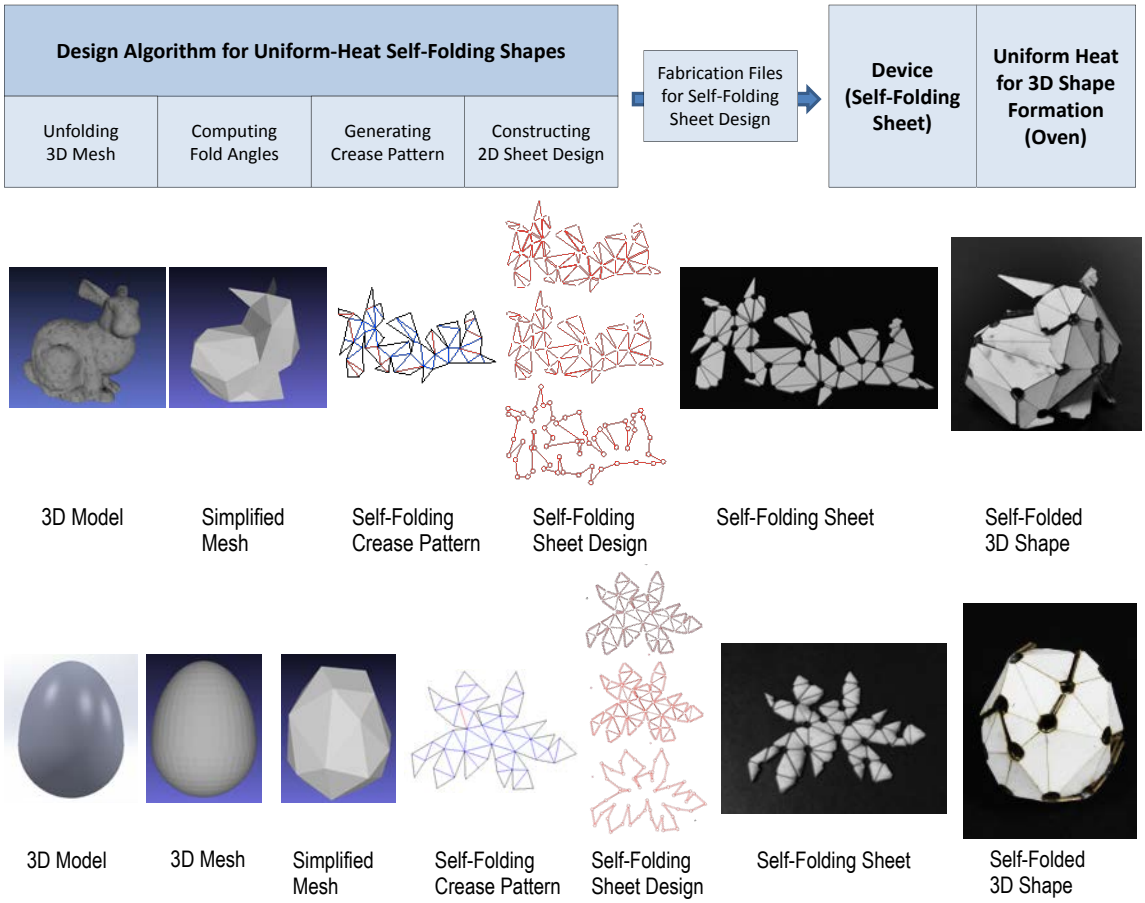


Fig. 2. Visual overview of the self-folding sheet development pipeline. The middle and bottom lines show the data transformation to develop a self-folding bunny and egg.

### A. Related Works

This paper builds on prior work in self-folding, computational origami and modular robots. Our previous work on creating self-folding devices controlling its actuators with an internal control system is described in [3]. In [4], [5] we discussed how to plan and program this type of self-folding sheets. [6], [7], [8], [9], [10], [11], [12] present other folding actuators and folding sensors controlled by internal electronic circuits.

Recently, various *self-folding actuators* triggered by external energy sources, such as heat [1], [2], light [13], or microwave [14], in both macro-scales and micro-scales [15] have been introduced. Since these types of actuators are activated by uniform external energy sources, a sheet containing these actuators does not require an internal control system. This simplifies the sheet design with respect to previous self-folding sheets [1], [2]. However, the automated design and control of these self-folding sheets arise as new challenges. We address these challenges with an algorithmic solution.

Previous work has addressed the generation of self-folding 2D DNA structures by controlling chemical bonding of DNAs (1D strings) without internal control systems [16]. By contrast, we construct 3D structures with 2D sheets composed of a few simple materials cut into geometries containing the self-folding information.

Theoretical work in computational origami and geometry has described various crease patterns for developing 2D/3D structures [17], [18], [19], [20], [21], [22] and robots [3], [23], [24], [4]. [25] describes a fabrication process of 3D micro-structures using manual folding.

Self-folding systems can be considered as a new family of modular systems, with tiles and hinges treated as basic modules (see [26] for a review of modular robotics). [27] proposes a programming method for self-folding systems, which they treat as many tiny cells that are smaller than the thickness of a sheet of paper. Each tiny cell is able to process a program, communicate to the other cells, and make small folding angles according to a given program. By contrast, the control information for the self-folding sheet described here is encoded in the design itself.

## II. PROBLEM FORMULATION

### A. Self-Folding Sheets Activated by Uniform Heating

A self-folding sheet is defined as a crease pattern composed of cuts and folding edges (hinges) as shown in Fig 3. A shape memory polymer (SMP) actuator is located along each folding edge of the sheet, and its fold angle is encoded

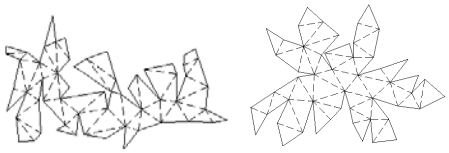

Fig. 3. Visualized self-folding crease pattern representing bunny and egg. The solid lines are cuts and the dash lines are edges (hinges). Each edge contains a fold angle.

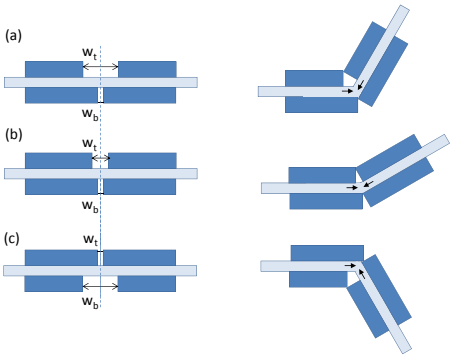


Fig. 4. Sandwiched actuator model. (Left) before activate. (right) after shrinking. The arrows show the shrinking directions.

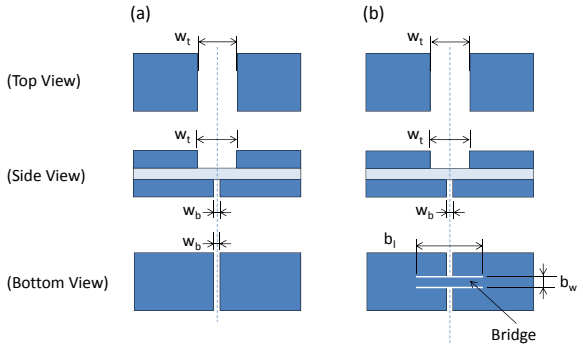


Fig. 5. Two types of actuators. (a) Basic-type actuator. (b) Bridge-type actuator.

by the geometry of the rigid material located at the edge. The fold angle is encoded in the design of each actuator. When uniform heat is applied to the sheet, all actuators fold their edges to their predefined fold angles simultaneously. Our previous work [1], [2] describes some designs that achieve this goal.

### B. Sandwiched Actuation Model for Self-Folding

The actuator is composed of three layers (Fig. 4). The top and bottom layers of the actuator are heat resistant materials (e.g. paper or Mylar). The middle layer is a SMP (e.g. prestrained polystyrene or polyvinyl chloride shrink film). Since all layers are strongly attached to each other, when the actuator is exposed to a uniform energy source, such as heat, light or water, a section of the uncovered middle layer shrinks, allowing the hinge to fold.

The spaces $w_t$, $w_b$ of the top and bottom layers determine folding angles and directions (Fig. 4). For example, if the gap of an actuator (a) is wider than the gap of another actuator (b), (a) folds to a greater extent. If the gap of the bottom layer is wider than the gap of the top layer, The actuator bends in the other direction (Fig. 4(c))

Fig. 5 shows two types of actuator design. The basic-type actuator has simple gaps on each top and bottom layer (Fig. 5(a), [2]). The bridge-type actuator has a simple gap on one side and a gap with a bridge on the other side (Fig. 5(b), [1]). Bridges hold object faces together during fabrication and reduce the number of release cuts required. The types of actuator design of self-folding sheets are determined by a selected actuator design function in Sec. IV-B.

## III. SELF-FOLDING SHEET DESIGN COMPILER

The *design compiling algorithm* converts a shape represented as a 3D *mesh*[1] shape or a 3D *origami design*[2] structure into a self-folding sheet design. Fig. 2 shows the steps of the algorithm and the development process for self-folding sheets: (1) unfolding a given 3D structure, (2) computing the fold angles, (3) constructing a 2D sheet crease pattern, and (4) constructing a 2D sheet design. Fabrication files for the sheet design are then output by the algorithm.

### A. Unfolding the 3D Shape

The objective of this algorithm is to compute the geometry of a 2D sheet that can be folded into the given 3D shape. Several algorithms exist to unfold 3D meshes or 3D origami designs [18], [28], [29]. Given a mesh, the compiling algorithm constructs a *net*[3] of the mesh on a plane without any collisions.

In this paper, a mesh is $M = (V,F)$ where $V$ is a finite set of the vertices and $F$ is a finite set of the faces. A net is $N = (V',E',F',T)$, where $V'$ is a finite set of the vertices, $E'$ is a finite set of the edges $e' = \{a,b\}$, $a,b$ are in $V'$, $F'$ is a finite set of the faces, $T$ is a finite set of $(e',t)$, and $t$ is a mark. $e(e') \in E(M)$ is an original edge of $e' \in E'$. $f(f') \in F(M)$ is an original face of $f' \in F'$. Since all vertices of a net are originally from a mesh, during the unfolding process, these tracking functions can be easily constructed.

Although all meshes, including meshes having holes, are unfolded on a plane, some meshes must be unfolded as a net with multiple disconnected groups of faces [30]. However, by tracking the origin of each edge, information for the connections between disconnected face groups can be accessed.

### B. Computing Fold Angles

The goal of this step is to compute the fold angles associated with all edges of a given mesh.

In origami theory [17], an edge (hinge) is a line segment between two faces. A *fold angle* of the edge is the supplement of the dihedral angle between two faces (Fig. 6 left). The sign of the fold angle is determined by the hinge: either a mountain fold or a valley fold (Fig. 6 right).

*Theorem 1:* Given a mesh, a finite set $U$ of all fold angles of the mesh are computed in $O(n^2 \times m)$ time and $O(n^2)$ space, where $n$ vertices and $m$ faces are in the mesh.

[1]A polygon mesh is a collection of faces that defines a polyhedral object.

[2]A origami design is a folded state of a paper structure encoded with a crease pattern and folded angles [5]

[3]A net of a mesh is an arrangement of edge-jointed faces in a plane.

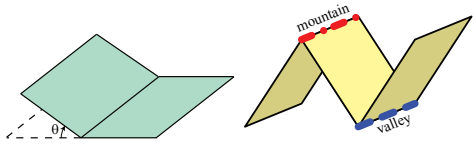


Fig. 6. (Left) The folding angle at a crease is the supplement of the dihedral angle. (Right) A crease can be folded as either a mountain fold or a valley fold.

*Proof:* For each edge, if the edge is not cut, there are two neighboring faces sharing the edge (Alg. 1 Step 1). By using the dot product and cross product of their normal vectors, the algorithm calculates the fold angle (Step b, c). Since there are at most $n^2$ edges, the algorithm computes and stores all angles in $O(n^2 \times m)$ time and $O(n^2)$ space. ■

**Computing Fold Angles**

1) Given a mesh $M = (V,F)$, where all the normal vectors of the faces point outside and the vertices of each face $(v_1, v_2, ..., v_k)$ are positioned counter-clockwise from top view.
2) For each edge $e = \{a,b\} \in E(M)$ where $e$ is not cut.
   a) Find two faces $f_1, f_2$ where $f_1$ contains directional edge $(a,b)$ and $f_2$ contains directional edge $(b,a)$.
   b) Get $u = acos(\frac{n_1 \cdot n_2}{|n_1||n_2|})$, where $n_1$, $n_2$ are the normal vectors of $f_1$, $f_2$, respectively.
   c) If $(a,b)$ and $n_1 \times n_2$ point to different directions, assign '-' to $u$; otherwise assign '+' to $u$.
   d) Insert $(e,u)$ into a finite set $U$.
3) Output $U$.

**Algorithm 1**: Algorithm to computing fold angles

### C. Constructing a Self-Folding Crease Pattern

The goal of this step is to take the 2D crease structure and the fold angles of a mesh as input and generate a crease structure that will self-fold the desired angles. Each edge in the original crease structure is thus mapped to a new crease structure capable of folding into the desired angle.

In this section, we show that given the crease structure and the fold angles of a mesh, the algorithm constructs a correct self-folding crease pattern (Thm. 2). Lem. 1 shows construction of a self-folding crease pattern, and Lem. 2 shows correctness of this crease pattern.

*Lemma 1:* Given a mesh $M$, its unfolding net $N$ and its finite fold angle set $U(M)$ (Sec. III-A, Thm. 1), Alg. 2 constructs a self-folding crease pattern in $O(n^2)$ times and space.

*Proof:* Given $M$, $N = (V',E',F',T)$, and $U(M)$, for each edge $e' \in E'$, Alg. 2 finds fold angle $u$ of its original edge $e(e')$ in $M$, and collects $u$ as a folding information $T'$. By replacing $T$ containing crease information (cut or hinge) to $T'$ containing desired angle information, Alg. 2 builds and outputs a self-folding crease pattern $(V',E',F',T')$ in $O(n^2)$ time and space. ■

**Self-Folding Crease Pattern Construction**

1) Given a mesh $M$, its net $N = (V',E',F',T)$ and a finite set $U(M)$ of $(e,u)$ where $e \in E(M)$ and $u$ is a fold angle.
2) For each $e' \in E'$, If $(e(e'),u) \in U$, then insert $(e',u)$ into $T'$, where $u$ is a fold angle.
3) For each $e' \in E'$, where $e'$ is a cut, then insert $(e', \langle cut \rangle)$ into $T'$.
4) Output self-folding crease pattern $(V',E',F',T')$ .

**Algorithm 2**: Algorithm to construct a self-folding crease pattern.

*Lemma 2:* Given a mesh $M$, if a self-folding crease pattern $N$ is generated by Alg. 2, $M'(N)$ is equal to $M$, where $M'(N)$ is the folded state of $N$

*Proof:* Let $L = \{f'_1, f'_2, ..., f'_k\}$, where $\exists e(e') = \exists e(e'')$, $e'$ is an edge of $f'_i$, $e''$ is an edge of $f'_j$, $j < i$, and $L = F'$. Let $L_p$ be $\{f'_1, f'_2, ..., f'_p\} \subseteq L$. Let $M'_t$ be $M(N_t)$. Let $F(M'_t)$ be $\{f''_1, f''_2, ..., f''_t\}$ where each $f''_i$ is a face of the folded state of $f'_i$.

For each $t \geq 1$, $P(t)$ is $M'_t = M_t$ where $L_t = F(N_t)$.
**Basis:** $P(1)$: $M'_1 = M_1$ because $f_1 = f''_1$.
**Induction step:** For each $k \geq 1$, we assume that $P(k)$ is true and we show that it is true for $t = k+1$.

The hypothesis states that $M'_k = M_k$, and $f_{k+1}$, $f''_{k+1}$ are the same shape. By the definition of $L_{k+1}$ $(= F_{k+1})$, $f'_{k+1}$ must be connected to $f'_s \in L_k$ and $f(f'_{k+1})$ is connected to $f(f'_s)$.

Let $u'$ be the fold angle of $e'$ between $f''_s$ and $f''_{k+1}$. Then $u = u'$ where $u$ is the fold angle of $e(e')$. Thus, $f_{k+1} = f''_{k+1}$ and $F(M'_{k+1}) = F(M_{k+1})$. Therefore $M'_{k+1} = M_{k+1}$ and $P(t)$ is true. ■

*Theorem 2:* Given $M$, $N$, and $U(M)$, Alg. 2 generates correct a self-folding crease pattern in $O(n^2)$ time and space, where $n$ is the number of the vertices.

*Proof:* Lemma 1 shows Alg. 2 builds a self-folding crease pattern in $O(n^2)$ time and space. Lemma 2 shows this crease pattern is correct. Therefore, Thm. 2 is true. ■

### D. Constructing a Self-Folding Sheet Design for Fabrication Files

This step constructs a self-folding sheet design by drawing all actuators of the sheet. Like an actuator composed of three layers, a self-folding sheet which is a cluster of the actuators is also composed of three layers. Fig. 7 shows an example design of a simple self-folding sheet.

An *actuator design* is $((w_t, w_c, w_b), b_l, b_w))$, where $w_t$, $w_c$ and $w_b$ are the gaps on the top, middle and bottom sheets, respectively, and $b_l$ and $b_w$ are the length and the width of the bridge (Fig. 5 (b)). If a variable of a design is ∋, then the algorithm skips the drawing of its layer. For example, if $w_c$ is ∋, the algorithm skips the drawing of the middle layer gap. If $b_l$ and $b_w$ are ∋, the actuator design does not have a bridge. $((w_t, ∋, w_b), (∋, ∋))$ and $((w_t, ∋, w_b), (b_l, b_w))$, respectively, represent the actuators in Fig. 5 (a) and (b).

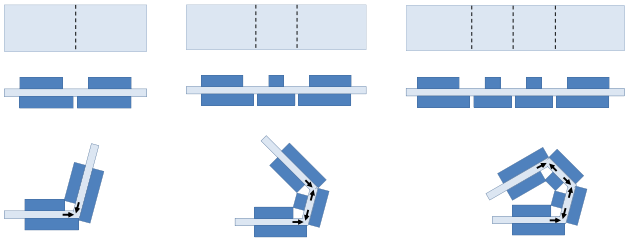

Fig. 7. Three examples of simple self-folding sheets embedding one, two and three actuators. The arrows show the shrinking directions.

**Constructing Self-Folding Sheet Design**

1) For each $(e,a) \in T$, where a given self-folding crease pattern is $(V,E,F,T)$ and $a$ is a fold angle ($a \notin \langle cut \rangle$).
   a) $d \leftarrow f(a)$, where $f$ is a given design mapping function and $d$ is an actuator design.
   b) If $(e,\langle cut \rangle) \notin T$:
      i) Draw $d$ on $e$ (Alg. 4).
   c) If $(e,\langle cut \rangle) \in T$:
      i) $w_{max} \leftarrow max(w_t(d), w_b(d))$.
      ii) Draw $((w_t(d), w_{max}, w_b(d)), (\ni, \ni))$ on $e$.
      iii) $T \leftarrow T - \{(e,\langle cut \rangle)\}$.
2) For each $(e,\langle cut \rangle) \in T$, draw $((0,0,0),(\ni,\ni))$ on $e$.
3) Output all actuator designs on the three layers as a self-folding sheet design.

**Algorithm 3**: Algorithm to construct a self-folding sheet design

Let $f : A \rightarrow D$ denote a *design mapping function*, where $A$ is a set of angles between $-180^\circ$ and $+180^\circ$ and $D$ is a set of actuator designs $((w_t, w_c, w_b), (b_l, b_w))$. Given a fold angle, we can draw the design of the three-layered actuator on three planes.

*Theorem 3:* A self-folding crease pattern has a valid self-folding sheet design, computable in $O(n^2)$ time and space, where $n$ is the number of the vertices.

*Proof:* A mesh has two types of edges: cuts and hinges. A net unfolded from a mesh also contains cuts and hinges but some cuts are originally from the hinges of the mesh. Alg. 3 draws the valid actuators for these edges. Step b draws actuators for the hinges in the net. Step c draws actuators for the cuts in the net which are originally the hinges in the mesh. Step 2 draws actuators for the cuts which were cuts in the mesh. Alg. 4 correctly draws each actuator on each layer of the self-folding sheet design. Each Step 1 and 2 of Alg. 3 runs in $O(n^2)$ time and space. ■

[4]Where $a = (x_a, y_a)$ and $b = (x_b, y_b)$, if $x_a \neq x_b$ then $\theta = atan(\frac{y_b - y_a}{x_b - x_a})$ and if $x_a = x_b$ then $\theta = 180^\circ$

[5]$c = (a+b) \times 0.5$

**Drawing Actuator**

1) Given an edge {a, b}, calculate the rotation angle[4] $\theta$ and the center point[5] $c$.
2) Given an actuator design, draw the $\theta$-rotated actuator design on $c$.

**Algorithm 4**: Algorithm to draw an actuator

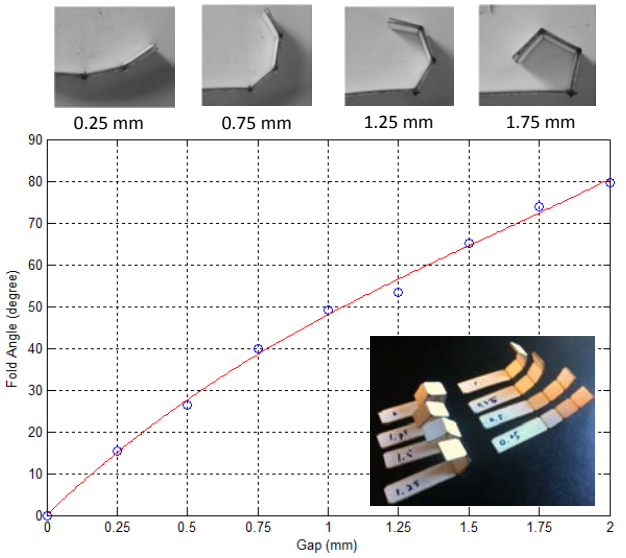


Fig. 8. Graph of an implemented actuator design function for the pin alignment process. The inset images show the test strips used to characterize the fold angle as a function of the size of the gap on the inner structural sheet.

## IV. ALGORITHM IMPLEMENTATION

### A. Software for Compiling the Printable 2D Design

We implemented the design algorithm (Fig. 2) in Java. The input file formats are Wavefront .obj for a 3D mesh and AutoCAD .dxf for a 3D origami design [5]. The output files are .dxf format.

To support various manufacturing processes of the self-folding sheet, the software supports script files to define the template of the fabrication files (outputs). Since we constructed self-folding sheets with two manufacturing processes, we built two template scripts for the folding alignment manufacturing process [1] and the pin alignment manufacturing process [2].

### B. Actuator Design Function

As started in Sec. III-D, given a fold angle, an actuator design mapping function $f$ outputs an actuator design.

*Definition 1:* A design mapping function is $f : A \rightarrow D$, where:
1. $A$ is a set of the angles ($-180^\circ \leq a \leq 180^\circ$),
2. $D$ is a set of the actuator designs $d = (w,b)$,
3. $S$ is a finite set of the fold angle samples $(a,d)$,
4. $(0,((0,\ni,0),b)) \in S$,
5. if $(a,d) \in S$, then $f(a) = d$, and
6. if $(a,d) \notin S$, then
$f(a) = (w(d_1) + \frac{a-a_1}{a_2-a_1} \times (w(d_2) - w(d_1)), b(d_1))$, where $a_1 < a < a_2$, $(a_1,d_1) \in S$, $(a_2,d_2) \in S$, $a_1 < a_3 < a_2$ and $(a_3,d_3) \notin S$.

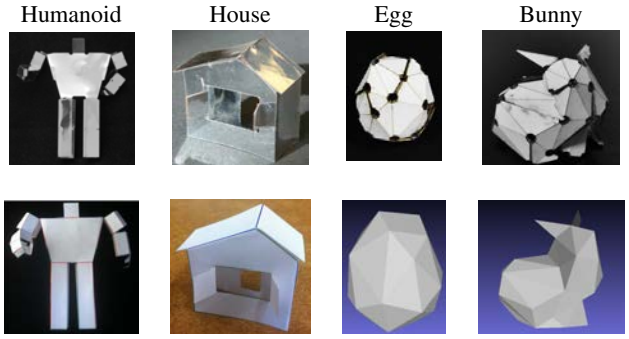


Fig. 9. (Top) Self-folded 3D shapes: the humanoid, house, egg and bunny shapes. (Bottom) Input models of the humanoid, house, egg and bunny. We modeled the humanoid and house designs with paper and coded them into origami designs. We modeled the egg and bunny shapes using CAD software.

Fig. 8 shows a graph of an implemented actuator design function. To characterize the fold angle as a function of the actuator geometry, we built eight self-folding strips with gaps on the inner layer in the range of $0.25mm$–$2mm$, and baked them at $170°C$. Each strip has three actuators with the identical gap dimensions. After baking, we measured the fold angle of each self-folded actuator. According to this graph, the actuator design function outputs the design of an actuator (5, 6 of Def. 1). We used an angle characterization method which we proposed in [2]. Since the strips are automatically designed by our pipeline, we can easily generate another set of strips for a different range of the gaps.

## V. EXPERIMENTS

We designed the experiments to evaluate the end-to-end pipeline for self-folding sheets. We built and baked a humanoid-shape and a house-shape with the folding alignment fabrication process [1], and egg and bunny shapes with the pin alignment fabrication process [2] (Fig. 9). We constructed the fabrication files with the design pipeline (Fig. 2) and the actuator design function of each fabrication process (Fig. 8).

### A. Design Pipeline

We built the humanoid and house origami shapes with paper and then coded the shapes into origami designs [5]. The 3D shape of the humanoid was composed of 41 faces and its 2D sheet contained 44 self-folding actuators (Tab. I). The 3D shape of the house was composed of 9 faces and its 2D sheet contained 8 actuators. Fig. 10 (a)(b) shows the fabrication files of the human shape and the house shape.

The egg shape was modeled in CAD software (Solidworks, Dassault Systemes SolidWorks Corp.), and exported as a 3D mesh with 2538 faces. We reduced the number of the faces to 50 using the MeshLab software [31], and then unfolded it with our software. The 2D sheet of the egg contained 48 actuators (Tab. I). We generated the fabrication files for the egg shape from this model. Fig. 10 (c) shows the fabrication files of the egg shape.

For the bunny shape, we downloaded the 3D Stanford Bunny (Rev 4, Stanford Computer Graphics Laboratory)

TABLE I
COMPLEXITY OF SHAPES

| | Humanoid | House |
|---|---|---|
| # of Faces | 41 | 9 |
| # of Actuators | 44 | 8 |
| Folding Range | -100.0°– 100.0° | -56.0°– 135.0° |
| | **Egg** | **Bunny** |
| # of Faces | 50 | 55 |
| # of Actuators | 48 | 54 |
| Folding Range | -0.6°– 55.0° | -103.4°– 67.1° |

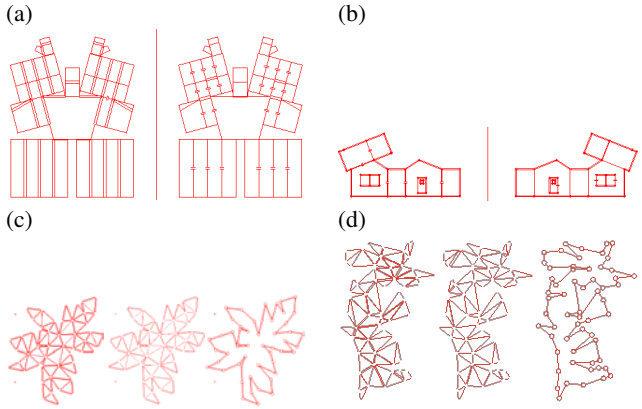


Fig. 10. (a)(b) Fabrication files of the folding alignment process generated for the humanoid and house. (c)(d) Fabrication files of the pin alignment process generated for the egg and bunny. (a)(b) Cut on the center guides the folding alignment while the top layer (right) and bottom layer (left) are sandwiched. (c)(d) Tiny holes are for the pin alignments. (Left) Cuts for the top layer. (Middle) Cuts for the bottom layer. (Right) Cuts for the all layers.

which contains 948 faces and reduced the number of faces to 55 using the MeshLab software. We unfolded this mesh and created the fabrication files with our software. Fig. 10 (d) is the fabrication files of the bunny shape.

### B. Self-Folding

After we built the fabrication files, we generated physical self-folding sheets for the humanoid, house, egg and bunny shapes. Fig. 11 shows two self-folding sheets built by the folding alignment process and the pin alignment process (Tab. II).

Each self-folding sheet was baked in an oven. We baked the humanoid and house at $55-65°C$ without preheating the oven (we put each sheet into the oven in room-temperature, and then increased the heat up to $65°C$). The egg and bunny were baked in an oven preheated to $120°C$. When we opened the oven to insert the sheet, the temperature dropped down to approximately $110°C$. While the sheet of the egg shape was placed on the preheated ceramic plate, the sheets of

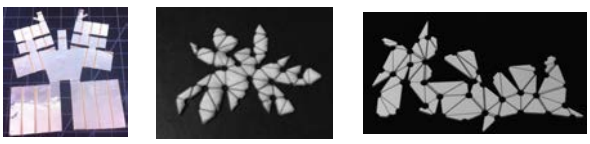

Fig. 11. Self-folding sheets (before bake) for humanoid (left), egg (center) and bunny (right)


Fig. 12. Frames from experiment of the self-folding humanoid shape by uniform heating. The sheet was built with the folding alignment process. The time elapsed since exposure to uniform heating is indicated in the higher-right corner of each frame (in minutes and seconds).

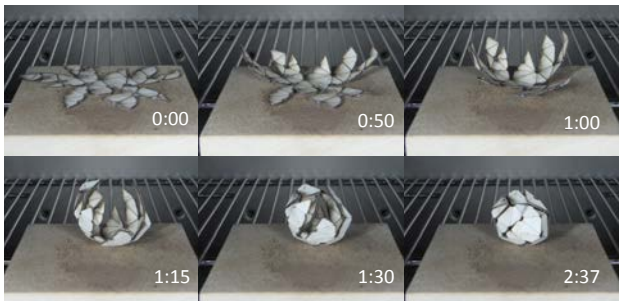


Fig. 13. Frames from experiment of the self-folding egg shape by uniform heating. The sheet was built with the folding alignment process. The time elapsed since exposure to uniform heating is indicated in the lower-right corner of each frame (in minutes and seconds).

the humanoid, house and bunny shapes were hung on the bars in the oven to reduce the effect of gravity on the self-folding process. Figs. 1, 12 and 13 show the frames of the experimental videos of the self-folding of the bunny, humanoid and egg shapes, respectively.

### C. Results and Discussion

Through our pipeline, we successfully constructed four self-folded structures. Tab. III shows the size of each shape before and after transformation.

Additionally, our approach designed and folded these structures rapidly (Tab. IV, each computing time is the average time of 10 runs). The computing time of each model was less than 0.5 seconds on a laptop. The self-folding time was also relatively short. All shapes folded themselves in under 7 minutes. Since the egg was folded on the preheated ceramic plate, it folded itself in 3 minutes.

The most time consuming step of the experimental design and fabrication of self-folding structures was the physical construction of the self-folding sheets. Since the design and folding steps are automated, these steps were finished in less than 7 minutes (Tab. IV). However, although we have clearly defined fabrication processes, because they still required some manual labor (CO2 laser machining, alignment, layer lamination, release cutting), took 2 - 3 hours to construct each self-folding sheet.

TABLE II
FABRICATION AND MATERIAL OF SELF-FOLDING SHEETS

| | Humanoid & House | Egg & Bunny |
|---|---|---|
| Fabrication Process | Folding | Pin |
| Folding Temp. | $55^{\circ}C$– $65^{\circ}C$ | $110^{\circ}C$– $120^{\circ}C$ |
| Top&Bottom Layers | Mylar | Paper |
| Middle Layer | PVC | PP |

TABLE III
SIZE OF SELF-FOLDED SHAPES

| | Humanoid | House |
|---|---|---|
| Sheet Size (mm) | $86 \times 112$ | $114 \times 69$ |
| Object Size (mm) | $71 \times 76 \times 27$ | $46 \times 38 \times 29$ |
| | Egg | Bunny |
| Sheet Size (mm) | $191 \times 171$ | $137 \times 82$ |
| Object Size (mm) | $31 \times 34 \times 26$ | $49 \times 43 \times 38$ |

TABLE IV
COMPUTING AND SELF-FOLDING TIMES

| | Humanoid | House |
|---|---|---|
| Computing Time | 478.17 ms | 392.17 ms |
| Folding Time | 4m 58s | 4m 57s |
| | Egg | Bunny |
| Computing Time | 478.2 ms | 464.5 ms |
| Folding Time | 2m 37s | 6m 26s |

| | |
|---|---|
| CPU | Intel Core i3-2350M (2.30GHz) |
| RAM | 4 GB |
| Storage | 500GB 5400rpm 2.5" HDD (TOSHIBA MK5076GSX) |
| Graphics | Intel HD Graphics 3000 |

The egg shape was successfully foleded on the plate (Fig. 13). Fig. 14 shows three eggs that were repeatedly constructed. The eggs (left, middle) were baked at $115^{\circ}C$, while the egg (right) was baked at $130^{\circ}C$. Although the front sides look almost identical, the back sides of the eggs have slight differences according to the temperature.

Since the bunny shape was heavy and complex, it did not have enough force to completely fold its lower extremities against gravity. We hung the bunny on the bar in the oven (Fig. 1). This improved the performance and the bunny was successfully folded (Fig. 15). However, more work is need to achieve repeatability.

While the house shape was underfolded, the humanoid shape was overfolded at the leg area. The reason for the overfolding in the humanoid's legs, but not the arms, is that the length of the edges along the legs was longer than along the arms but the size of the associated bridges was constant.

## VI. CONCLUSION AND FUTURE WORKS

In this paper, we explored and analyzed an end-to-end approach to making self-folding sheets activated by uniform-heat. We introduced a design pipeline which automatically generates folding information, then compiles this information into fabrication files. We proposed the self-folding sheet design algorithm and proved its correctness. We also demon-

Fig. 14. Front (top) and back (bottom) sides of three self-folded eggs. (left, middle, right).

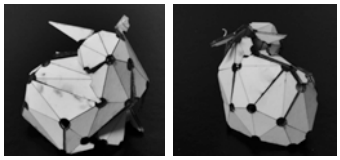

Fig. 15. Front and back sides of the self-folded bunny.

strated the implementation of this pipeline and characterized the actuator design function to convert the theoretical design to a physical self-folding sheet. Finally, we demonstrated this approach experimentally by generating self-folding sheets for the fabrication of four target shapes. The pipeline correctly designed and built the sheets. The experiments were successful, and the sheets were folded themselves in relatively short times when exposed to uniform heating.

Some practical challenges remain to be addressed in the physical fabrication of self-folding sheets. Delamination of the SMP layers from the structural layers occurred near the edges of our self-folding sheets for the egg and bunny shapes. This may be mitigated by sealing the edges of the sheet or with improved adhesion.

Other challenges are the evaluation of self-folding sheets. Although the back side of the bunny shape in Fig. 15 shows the completion of the shape, it was hard to evaluate or analyze the completeness of the self-folded model. The development of benchmark and evaluation methods for self-folding sheets would support a systematic approach to improve the self-folding.

During self-folding, the collisions of the faces were a problem. Since we did not have a simulator or optimizer, to avoid the collisions, particularly for the bunny unfolding, we rearranged some faces. One solution would be the development of the self-folding simulator to minimize the collision while the pipeline generates the design. Another solution would be an upgraded design algorithm to allow the self-folding sheet to fold itself with multiple folding steps.

## REFERENCES


[1] S. Miyashita, C. Onal, and D. Rus. Self-pop-up cylindrical structure by global heating. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (to appear)*. IEEE, 2013.

[2] M. Tolley, S. Felton, S. Miyashita, L. Xu, B. Shin, M. Zhou, D. Rus, and R. Wood. Self-folding shape memory laminates for automated fabrication. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (to appear)*. IEEE, 2013.

[3] E. Hawkes, B. An, N. Benbernou, H. Tanaka, S. Kim, E. Demaine, D. Rus, and R. Wood. Programmable matter by folding. *Proceedings of the National Academy of Sciences*, 107(28):12441–12445, 2010.

[4] B. An and D. Rus. Designing and programming self-folding sheet. *Robotics and Autonomous Systems (to appear)*, 2013.

[5] B. An, N. Benbernou, E. Demaine, and D. Rus. Planning to fold multiple objects from a single self-folding sheet. *Robotica, Special Issue on Robotic Self-X Systems*, 29(1):87–102, January 2011.

[6] S. Felton, M. Tolley, B. Shin, C. Onal, E. Demaine, D. Rus, and R. Wood. Self-folding with shape memory composites. *Soft Matter*, 9:7688–7694, 2013.

[7] S. Felton, M. Tolley, C. Onal, D. Rus, and R. Wood. Robot self-assembly by folding: A printed inchworm robot. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013.

[8] J. Paik, E. Hawkes, and R. Wood. A novel low-profile shape memory alloy torsional actuator. *Smart Materials and Structures*, 19(12):125014.

[9] J. Paik and R. Wood. A bidirectional shape memory alloy folding actuator. *Smart Materials and Structures*, 21(6):065013, 2012.

[10] E. Smela, O. Ingans, and I. Lundstrm. Controlled folding of micrometer-size structures. *Science*, 268(5218):1735–1738, 1995.

[11] E. Smela. A microfabricated movable electrochromic "pixel" based on polypyrrole. *Advanced Materials*, 11(16):1343–1345, 1999.

[12] J. Paik, R. Kramer, and R. Wood. Stretchable circuits and sensors for robotic origami. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 414–420, 2011.

[13] Y. Liu, J. Boyles, J. Genzer, and M. Dickey. Self-folding of polymer sheets using local light absorption. *Soft Matter*, 8:1764–1769, 2012.

[14] K. Yasu and M. Inami. Popapy: instant paper craft made up in a microwave oven. In *The 9th international conference on Advances in Computer Entertainment*, 2012.

[15] N. Bassik, G. Sttern, and D. Gracias. Microassembly based on hands free origami with bidirectional curvature. *Applied Physics Letters*, 95:1764–1769, 2012.

[16] P Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

[17] E. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, July 2007.

[18] T. Tachi. Origamizing polyhedral surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):298–311, 2010.

[19] N. Benbernou, E. Demaine, M. Demaine, and A. Ovadya. A universal crease pattern for folding orthogonal shapes. arXiv:0909.5388, September 2009.

[20] E. Demaine, S. Devadoss, J. Mitchell, and J. O'Rourke. Continuous foldability of polygonal paper. In *the 16th Canadian Conference on Computational Geometry (CCCG'04)*, pages 64–67, August 2004.

[21] E. Demaine, M. Demaine, and J Ku. Folding any orthogonal maze. In *Proc. fifth international meeting of origami science, mathematics, and education*, pages 449–55, 2011.

[22] E. Demaine, S. Fekete, and R. Lang. Circle packing for origami design is hard. In *Origami$^5$: Proceedings of the 5th International Conference on Origami in Science, Mathematics and Education (OSME 2010)*, pages 609–626. A K Peters, Singapore, July 13–17 2010.

[23] C. Onal, R. Wood, and D. Rus. An origami-inspired approach to worm robots. *IEEE/ASME Transactions on Mechatronics*, 18(2):430–438, 2013.

[24] D. Soltero, B. Julian, C. Onal, and D. Rus. A lightweight modular 12-dof print-and-fold hexapod. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (to appear)*. IEEE, 2013.

[25] S. Brittain, O. Schueller, H. Wu, S. Whitesides, and G. Whitesides. Microorigami: Fabrication of small, three-dimensional, metallic structures. *Journal of Physical Chemistry B*, 105(2):347–350, 2001.

[26] M. Yim, Wei-Min Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *Robotics & Automation Magazine, IEEE*, 14(1):43–52, 2007.

[27] R. Nagpal. Programmable self-assembly using biologically-inspired multiagent control. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 418–425, July 2002.

[28] S. Takahashi and C. Lin H. Yen H. Wu, S. Saw. Optimized topological surgery for unfolding 3d meshes. *Computer Graphics Forum*, 30:2077–2086, 2011.

[29] T. Tachi. Simulation of rigid origami. In *Origami$^4$: Proceedings of 40SME*, pages 175–187, 2009.

[30] M. Bern, E. Demaine, D. Eppstein, E. Kuo, A. Mantler, and J. Snoeyink. Ununfoldable polyhedra with convex faces. *Computational Geometry: Theory and Applications*, 24(2):51–62, February 2003. Special issue of selected papers from the 4th CGC Workshop on Computational Geometry, 1999.

[31] Meshlab. WebSite: http://meshlab.sourceforge.net/.