Sticker Controller and Sticker Programming for Smart Sheets (Self-Folding Sheets)

by

Byoungkwon An

B.S. in Physics, Soongsil University (2004)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2011

Author ....
Department of Electrical Engineering and Computer Science
September 5, 2011

Certified by ....
Daniela Rus
Professor
Thesis Supervisor

Accepted by ....
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Theses

Sticker Controller and Sticker Programming for Smart Sheets (Self-Folding Sheets)

by

Byoungkwon An

B.S. in Physics, Soongsil University (2004)

Submitted to the Department of Electrical Engineering and Computer Science
on September 5, 2011, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

This thesis describes a self-folding sheet that is capable of origami-style autonomous folding. We describe the hardware device we designed and fabricated. This device, called a self-folding sheet, is a sheet with a box-pleat pattern and an integrated electronic substrate and actuators. The sheet is programmed and controlled using a new idea called sticker programming. We describe the architecture of a machine that can be programmed by sticker programming and its instantiation. We also describe planning algorithm and automatic programming algorithm for controlling a given sheet to self-fold into a desired shape. Finally we present experiments with a 4×4 hardware device and an 8×8 hardware device.

Thesis Supervisor: Daniela Rus
Title: Professor

Acknowledgments

I thank to Prof. Daniela Rus for invaluable discussion, insights, comments and supports.

I thank to Sookee Choi for invaluable comments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A smart sheet (called the self-folding sheet) is a robotic sheet that autonomously transforms its shape by folding into the users' desired shapes. Our vision is to develop the hardware and software technology that will allow users to makes shapes by starting with a self-folding sheet and adding physical stickers to select and trigger a self-folding control sequence guaranteed to achieve the desired shape. We imagine sheets capable of folding as a variety of objects, such as a table, an airplane, or a tent. Applications include digital fabrication, on-demand construction of objects in remote environments, on-demand creation of tools, etc. Figure 1-1 shows examples of complex 3D shapes created manually by origami (folding) from one piece of paper. We aim to automate the creation of origami objects.

Figure 1-1: Examples of origami figures each folded with one paper: Ryujin 3.5[20] (left), Gundalf[18], Aragorn[17], Legolas[19] (characters of the Lord of the Rings) (right).

In this thesis we explore the programming aspects of self-folding sheets (smart sheets) by a novel concept called sticker programming. We will investigate the hardware and algorithms needed for an easy selection of the desired shape, when the self-folding sheet can achieve multiple shapes. The key technical challenges include:

- How to design a self-folding sheet capable of making multiple shapes.
- How to program the self-folding sheet to achieve a desired shape.
- How to control the execution of the desired shape.

In our previous work [14] we demonstrated algorithms that encompass design and control for a self-folding sheet to achieve two shapes and showed experimentally a sheet that can make a boat and a plane. This work does not consider the programming of the sheet. The sheet is created with the box pleat structure [14, 7]. There are actuators on each edge which are activated by applying power to them.

Our other previous work [2] focused on the origami planning that automatically selects the sequence of edge activations to achieve the desired shape. This previous algorithm does not cover the programming or controlling the self-folding sheets.

1.1 Approach

We explore sticker controller and sticker programming as new approaches to designing and programming self-folding sheets. Figure 1-2 shows the overview of our approach. The sticker controller is a machine that controls self-folding sheets to achieve the user's goal shapes. Just like a micro processor runs machine codes, the sticker controller runs executable stickers. Executable stickers are physical devices that can be added to the self-folding sheet to trigger the control sequence that achieves a desired goal shape. Just like the processor controls the devices of the PC, the sticker controller controls a self-folding sheet.

Sticker programming is a new programming paradigm for the sticker controller. When we give k desired shapes as inputs to the planner, it automatically generates an origami-like folding plan for the shapes. Then, the sticker compiler (compiling algorithm) and the

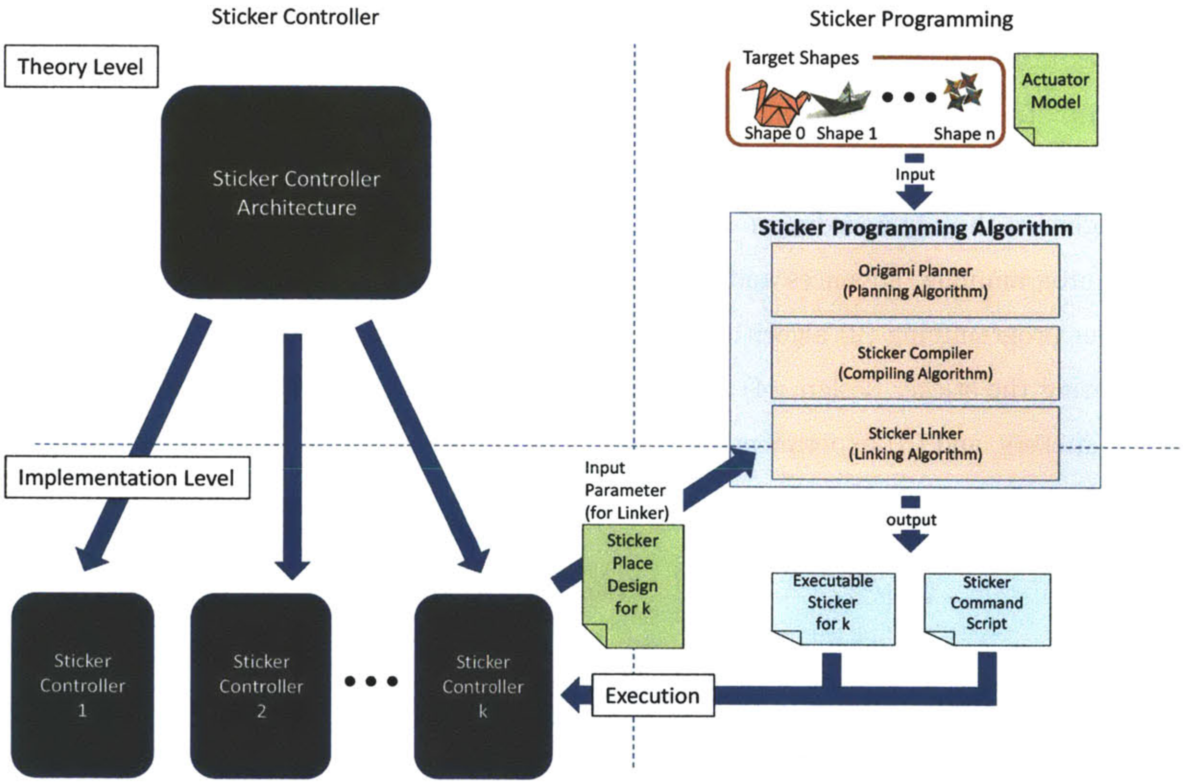


Figure 1-2: Overview of the approach for programming and controlling the self-folding sheet: the sticker controller and the sticker programming.

sticker linker (linking algorithm) automatically generate the executable sticker design for the input shapes. We develop an executable sticker considering to the executable sticker design. The executable sticker is analogous to a diskette for the Mac or PC. When we input the executable sticker to the sticker controller, the sticker controller controls the self-folding sheet.

The sticker controller can be manufactured using various types of materials (using electrical, hydro, heat, or etc. energy) and scales (from pico to macro). Figure 1-3 shows three examples manufactured with one sticker controller architecture but different material and processes. To keep the 2 dimensional formation of the system, the sticker computer must not have any directional parts (e.g. diode or transistor). The architecture also needs to be designed with only non-directional parts, such as wires or tubes. To achieve the goal of sticker programming, advances and innovation are needed for (1) designing devices capable of sticker programming (2) algorithms for automating the sticker design, (3) algorithms for automating the sticker design, (4) algorithms for automating the origami control triggered by the stickers, and (5) experiments.

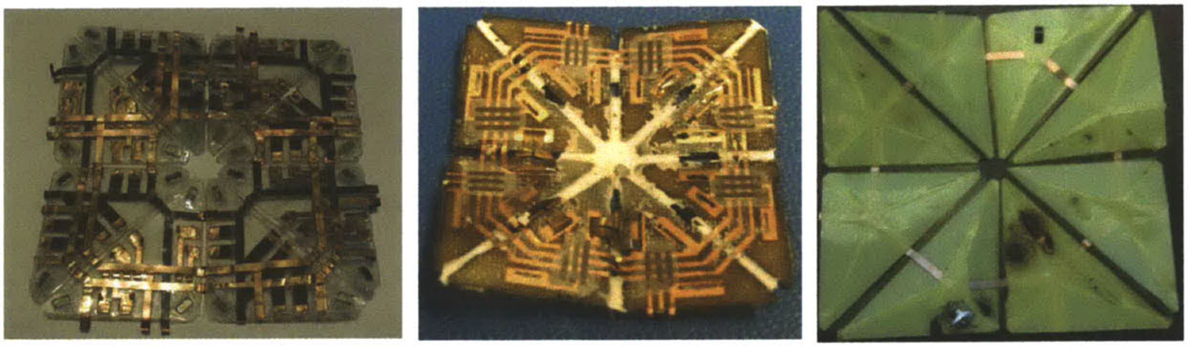


Figure 1-3: Three examples of manufactured 2 × 2 self-folding sheets. Each device was created using the same architecture but a different manufacturing process

1.2 Organization of the Thesis

In this thesis, we introduce the related works in Chapter 2. We introduce the problem formulation in Chapter 3. We describe the sticker controller architecture in Chapter 4. We develop the sticker programming development process and algorithm in Chapter 5. We build two different implementations of the hardware devices and experiment them with the sticker programming in Chapter 6. We discuss conclusions and future works in Chapter 7.

1.3 Contributions

This thesis makes the following contributions.

- the concept of programming a self-folding sheet using the notion of sticker programming
- an algorithm for automatically designing actuator placement for self-folding sheet
- an algorithm for automatically designing stickers that can program a desired shape
- two self-folding planners for archiving one shape and multiple shapes with a self-folding sheet
- two hardware prototypes that implement self-folding sheets
- experiments with the 4×4 self-folding sheet capable of automatically generating 2 simple shapes.
- experiments with the 8×8 self-folding sheet capable of generating 2 complex shapes

Chapter 2

Related Work

Our work on self-folding sheets builds on important prior work in modular self-reconfiguring robots and the study of origami.

2.1 Self-Folding Sheet

Self-folding sheets are a type of self-reconfiguring robot systems [14, 2]. The self-folding sheet system's topology is a square sheet. This sheet has associated control that actuates its edges in the correct sequence to achieve a desired shape. Figure 2-1 shows a self-folding sheet achieving a boat transformation in simulation and physical experiments.

More specifically, self-folding sheets consist of triangular tiles connected by flexible hinges arranged in an m × n box-pleated pattern. The tiles are made from rigid material, while the hinges are flexible. Hinges can be folded by actuators, which occupy some space either within a triangle or along a hinge. Figure 1-3 shows examples of manufactured self-folding sheets. These examples use different materials for the sticker but they are in the same topology.

2.2 Self-Reconfigurable System and Algorithm

Most of prior research in the field addresses the design of modular self-reconfigurable systems [1, 23, 29, 25, 27, 26, 24, 21, 15, 35, 31, 28, 36, 37, 16, 40, 34, 33, 10, 32, 38] and related

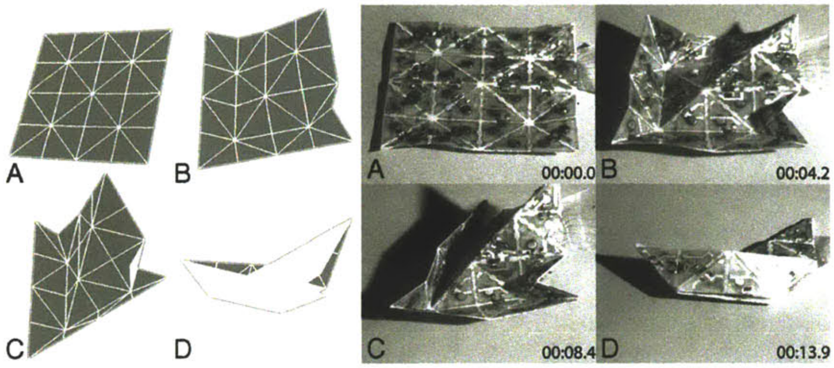


Figure 2-1: Simulation (left) and experiments (right, with time shown in lower right) of a self-folding boat (A). All actuators receiving current (B). Immediately before magnetic closures engage (C). Finished boat on side (D). [14]

shape-planning algorithms [3, 8, 9, 23, 29, 25, 27] ; for example: An's "EM-Cube" [1]; Kotay and Rus's "Molecule" [24], [22]; Rus and Vona's "Crystalline" [35]; Murata, Kuokawa, et al.'s "3D Fracta"[33]. Other self-reconfiguration systems and algorithms include [31, 28, 36, 37, 16, 40, 34]. Prior or ongoing works that use cube-shaped modules include Gilpin, Kotay et al.'s "Miche" [13], Koseki, Minami, et al.'s "HOBIE"[21], White, Zykov, et al's self-assembly system [38], and Unsal, Kiliccote et al's "I-Cube" [25].

The self-folding sheet is different than modular self-reconfiguring robots in that the modules in a self-reconfiguring system are disconnected, while the self-folding sheet has a mesh of connected tiles, each tile serving the role of a module.

2.3 Origami Theory

To develop the programming method for the self-folding sheet, we build on the new theory of universal crease patterns. Our theoretical model for the self-folding sheet has a box-pleat pattern 2-2. A box-pleat pattern is composed of the tiles ad joints. The tiles are isosceles right triangles made from rigid material. The joints are placed on the edges of the sheets made from flexible material. Demaine and et al. [7] recently proved that an $n \times n$ box-pleat tiling has as a folded state any polyhedral surface made up of $O(n)$ unit cubes on the cubic lattice. They [12] also showed that any such folded state can be reached by a continuous

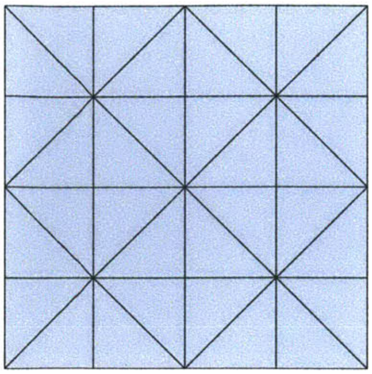


Figure 2-2: Box-pleated crease pattern

folding motion without the material penetrating itself. With these theories, the box-pleat crease pattern allows the creation of exponentially many foldings out of a single tiled crease pattern.

2.3.1 Robotic Origami Folding

Prior work on robotic origami folding considered the design of robot that fold the sheet into a folded structure and supporting algorithms. Balkcom and Mason [5, 6, 4] have built a robot that makes a sequence of *simple folds*—folds along a single line at a time. The robot folds a restrictive class of origami models. By contrast, our folds are generally more complicated, involving several simultaneous creases. Many other works considered robots for automatic folding of cartons and packaging [30, 11]. All of the prior origami robots manipulate the object to fold it with the external actuation. By contrast, in our work, the actuation of the sheet is internal; the sheet itself is a self-folding robot and the self-folding robot transforms itself into the target object.

Chapter 3

Problem Formulation

This chapter discusses our model for self-folding sheets and the self-folding control formulation.

3.1 Model of Self-Folding Sheet

A self-folding sheet is a box-pleated 2-dimensional sheet designed to transform itself into the desired shapes by folding selected edges. Figure 3-1 shows the simplified structure of the 4 × 4 Self-Folding Sheet. The kinematic components of the sheet include tiles, joints (hinges), and actuators. The controlling components include a sticker controller and sticker programs. We describe in the controlling components in Chapters 4 and 5.

The tiles are isosceles right triangles made from rigid material. They work as the structure of the self-folding sheet. The joints are placed on the edges of the sheets. In this model, the tiles and the joints follow the ideal tile models and joints models that are typically used in computational geometry. In the ideal model, the tile is a 2-dimensional ridged material and does not have any thickness. The joint is an 1-dimensional line that connects the tiles. The tiles locally rotate (fold) around its joint. The dihedral angle is in the range 0° to ±180°.

The folds angle is the supplement of the dihedral angle between the two face meeting hinge, as shown in Figure 3-2. The sign of the fold angle determines the crease as either a

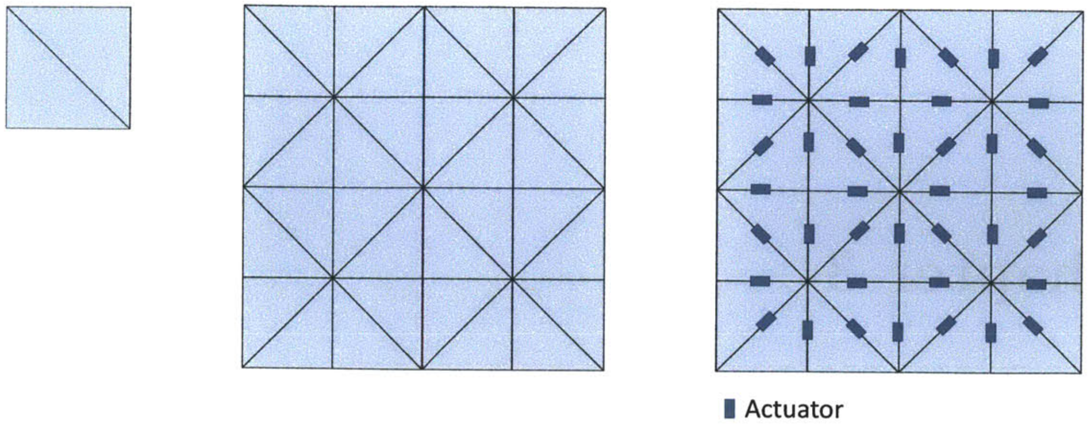


Figure 3-1: An 1×1 self-folding sheet (left) is a basic unit of self-folding sheets. A 4×4 self-folding sheet (middle), and with folding actuators (right). The 4×4 sheet is composed of 16 ($= 4 \times 4$) 1×1 self-folding sheets

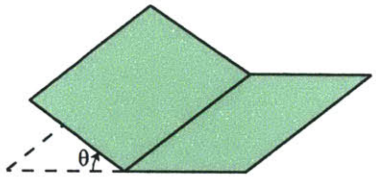


Figure 3-2: The fold angle at a crease is the supplement of the dihedral angle.[2]

mountain fold or a valley fold, as shown in Figure 3-3. We use red lines to indicate mountain folds and blue lines to indicate valley folds.

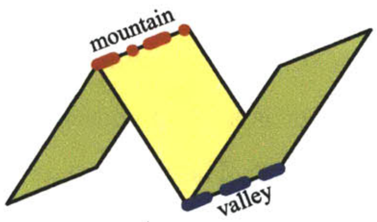


Figure 3-3: A crease can be folded as either a mountain fold (left) or a valley fold (right).[2]

3.1.1 Actuation Model

The actuators fold the sheet's edges. The actuators of most of our prior manufactured self-folding sheets are made of shape memory alloy (SMA) springs or SMA sheets. When the SMA actuators reach a particular temperature, they go to their memorized shape. Our model can be instantiated with other types of actuators that use alternative energy sources, such as the water pressure or piezoelectric [39].

In *the actuation model*, an actuator can fold the edges to the finite number of the angles. When the actuator received the signal, it goes to the angles according to the signal. The actuator is formally defined as follows:

Definition 3.1.1 *The folding actuator, FA, is expressed with 3-tuple, (Σ, A, δ), where:*
Σ is a finite set of the actuator code.
A is a finite set of the folding angles.
δ is an angle function, that is, $\delta : \Sigma \rightarrow A$

For example, if there is an actuator that goes to 0°, +90°, -90°, +180°, or -180° [1]. We can express the actuator FA = (Σ, A, δ), where:
Σ = { 0000, 0001, 0010, 0100, 1000 }
A is { 0°, +90°, -90°, +180°, -180° }
$\delta(0000) = 0°$
$\delta(0001) = +90°$
$\delta(0010) = -90°$
$\delta(0100) = +180°$
$\delta(1000) = -180°$

Y-shape actuators are used for our experiments (Fig. 3-4). The Y-shape actuator is a one-directional one-angle actuator, which folds to 180° only. To fold an edge to the angle +180°, we place the actuator on the bottom side of a self-folding sheet. To fold an edge to

[1] The airplane, box and piano are also composed of the finite number of angles, such as 0°, +90°, -90°, +180°, or -180°

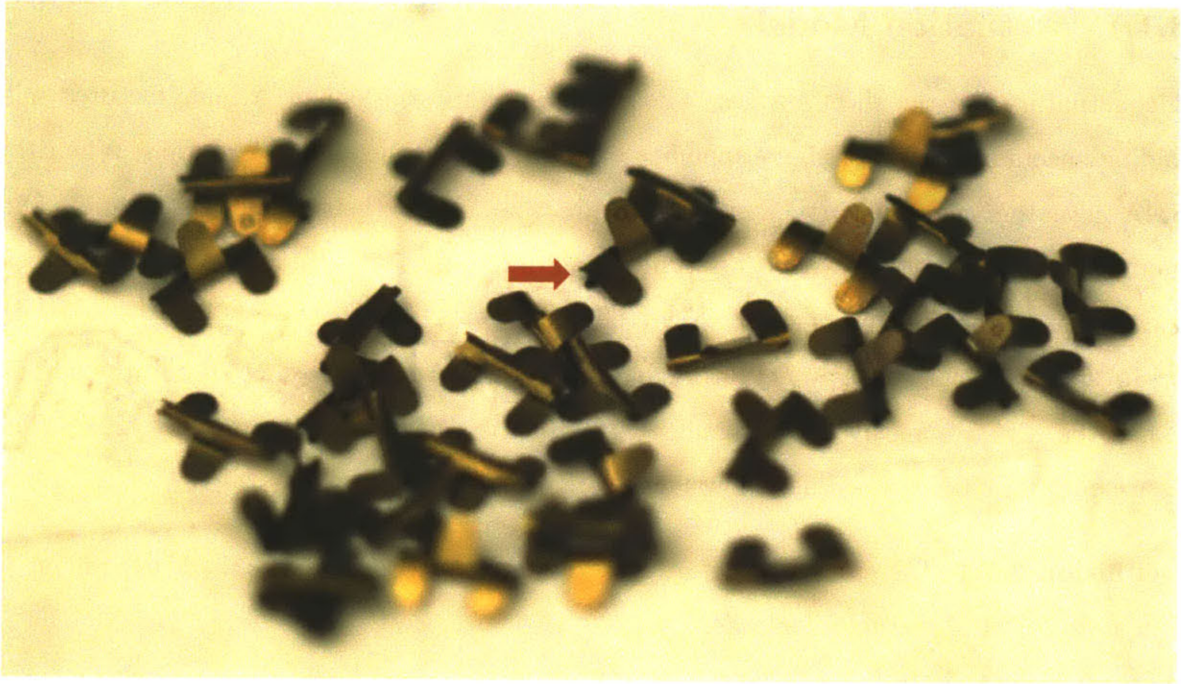


Figure 3-4: Y-Type Actuators. It is an one-directional one-angle actuator. The red arrow points a loop of an actuator.

the angle -180°, we place the actuator on the top side of a self-folding sheet. The model for the Y-shape actuator is YA = (Σ, A, δ):

$\Sigma = \{ 0, 1 \}$

A is $\{ 0°, \pm 180° \}$

$\delta(0000) = 0°$

$\delta(0001) = \pm 180°$

3.1.2 1 × 1 Self-Folding Sheet

An 1 × 1 self-folding sheet is a square shape with one diagonal hinge, as shown in Figure 3-1 (left). It is the basic unit of the self-folding sheet.

3.1.3 m × n Self-Folding Sheet

If the self-folding sheet is composed of the m columns and n rows 1 × 1 self-folding sheet, we call it $m \times n$ self-folding sheet.

A $m \times n$ self-folding sheet is composed of $m \times n$ rows of 1 × 1 self-folding sheets. For example, A 4 × 4 self-folding sheet is composed of 16 1 × 1 self-folding sheets (Figure 3-1).

An $m \times n$ self-folding sheet has $3mn - (m + n)(= mn + m(n - 1) + n(m - 1))$ joints. Because each 1 × 1 self-folding sheet of the $m \times n$ sheet has one diagonal joint, the number of the diagonal joints is mn. Because each column of the sheet has $(n-1)$ vertical joints, the number of total vertical joints is $m(n-1)$. The number of total horizontal joints is $n(m-1)$.

Controlling an $m \times n$ self-folding sheet is very challenging because there exist many possible combinations for actuating motions.

Chapter 4

Sticker Controller Architecture

The sticker controller is a module that contains the electronic substrate required to fold the self-folding sheet into users' desired shapes, when the users provide sticker programs to the controller. It provides the user with a programming interface which is implemented using physical materials.

Figure 4-1 shows two examples of self-folding sheets that include sticker controllers. The sheets are manufactured with different materials and processes. The details fabrication are discussed in Chapter 6. In this chapter, we will discuss the sticker controller architecture.

4.1 Sticker Controller

Figure 4-2 shows an example sticker controller for the 4×4 self-folding sheet. An $m \times n$ sticker controller is composed of m columns and n rows of 1×1 sticker controllers.

Figure 4-3 shows the model for the 4×4 sticker controller. The sticker controller is composed of a signal interface, a circuit, (actuator) sockets, and sticker controller units (sticker places).

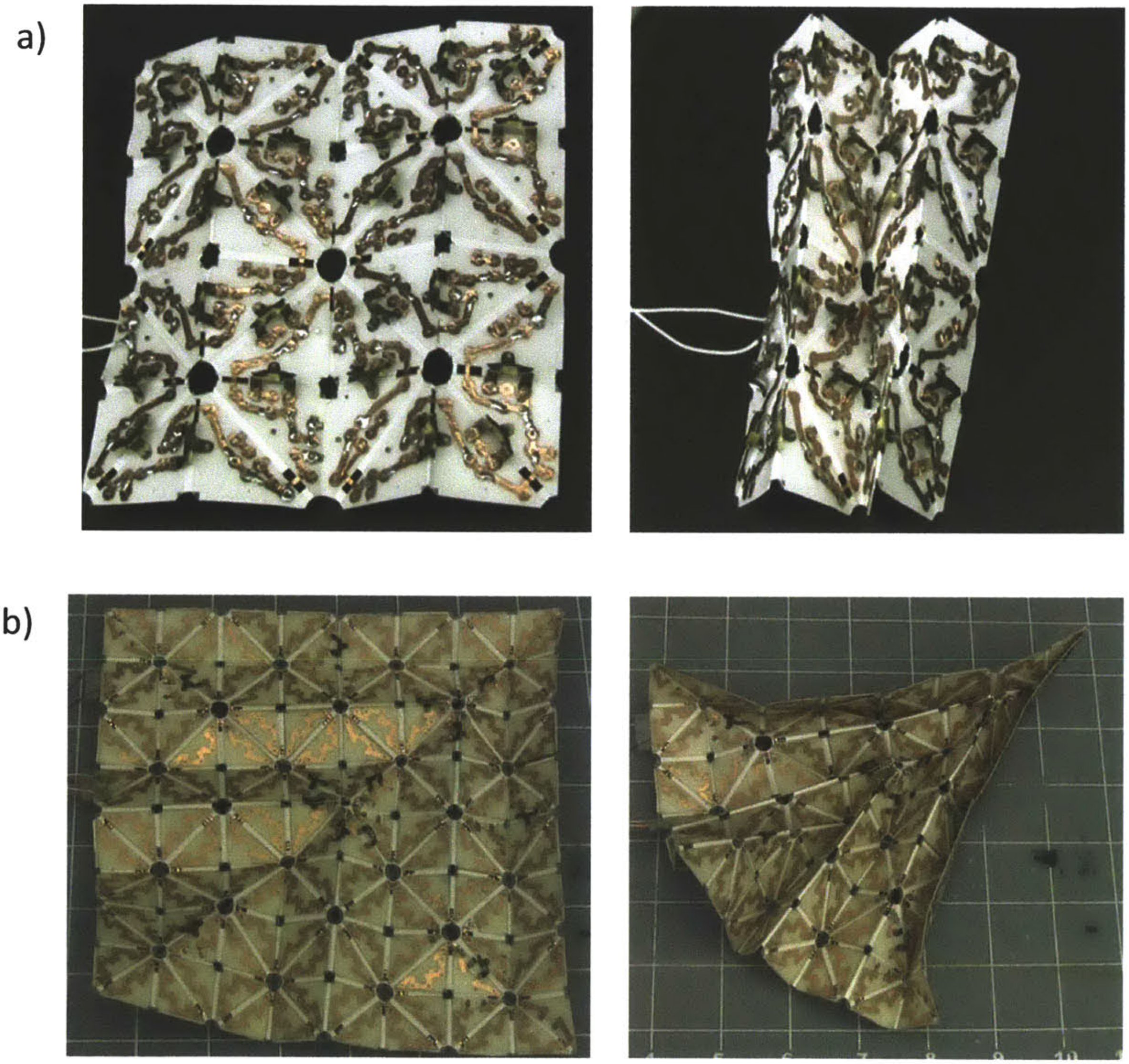


Figure 4-1: Two examples of different implementations of self-folding sheets (Ch. 6). (a) 4×4 self-folding sheet with the vertical folding program. The sheet is of size $96mm \times 96mm$ and uses copper tape. (b) 8×8 self-folding sheet with the space shuttle-like shape folding program. The sheet is of size $192mm \times 192mm$ and uses copper foil and PEEK. Each device was created using the same architecture but a different manufacturing process

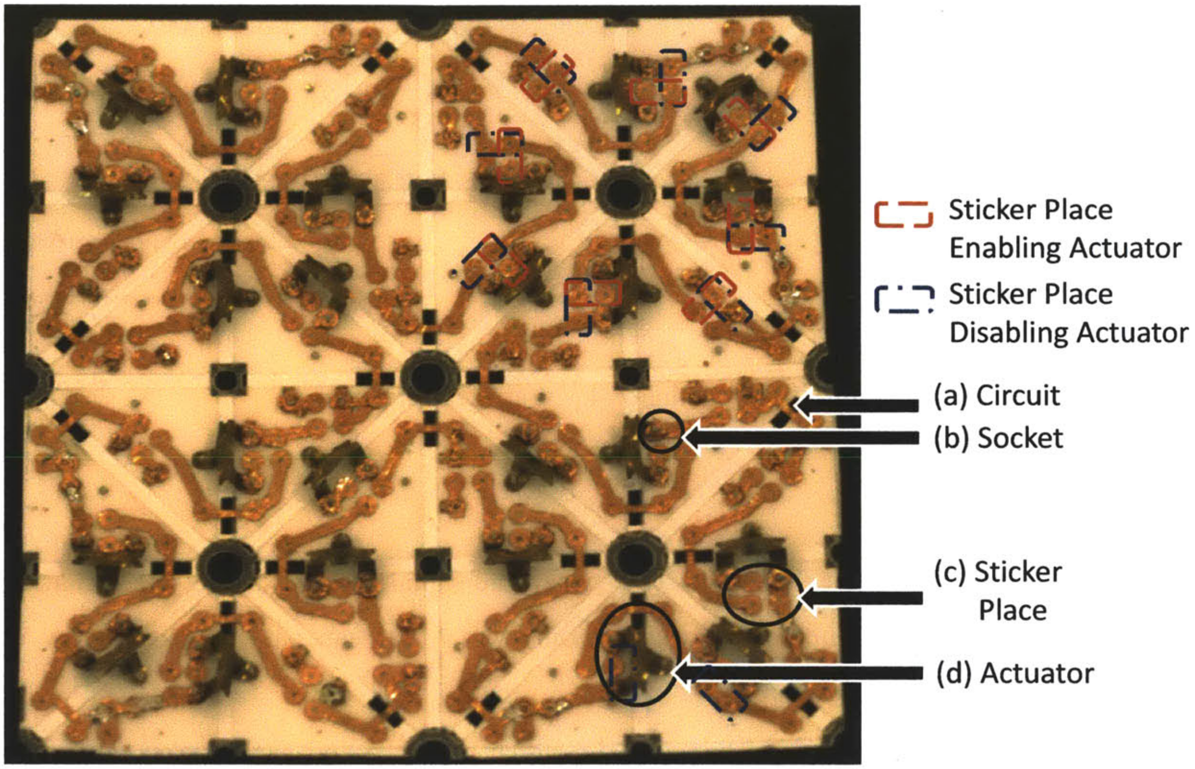
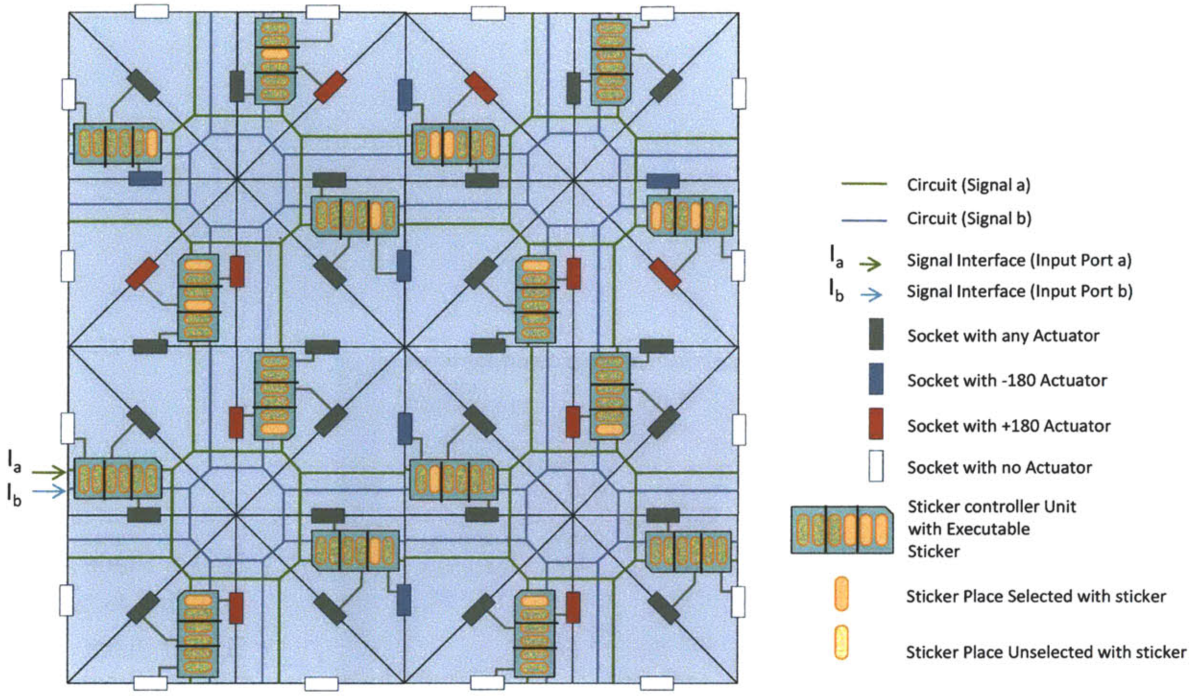

Sticker Place Enabling Actuator
Sticker Place Disabling Actuator
(a) Circuit
(b) Socket
(c) Sticker Place
(d) Actuator

Figure 4-2: The 4 × 4 sticker controller for the self-folding sheet experiment described in Chapter 6.



Ia
Ib
Circuit (Signal a)
Circuit (Signal b)
Ia Signal Interface (Input Port a)
Ib Signal Interface (Input Port b)
Socket with any Actuator
Socket with -180 Actuator
Socket with +180 Actuator
Socket with no Actuator
Sticker controller Unit with Executable Sticker
Sticker Place Selected with sticker
Sticker Place Unselected with sticker

Figure 4-3: Model of 4 × 4 sticker controller

4.1.1 The Circuit

The circuit is a network that distributes the energy and signals for controlling the actuating units (Fig. 4-3). The circuit is composed of only wires that can pass enough energy to activate all actuators simultaneously.

The circuit makes connections between the signal interface and each sticker controller unit, and each sticker controller unit and three sockets.

In our model, we use a *parallel circuit* and eclipse the ground layer of the circuit. With the parallel circuit, each part (sticker controller unit and socket) is parallely connected to the circuit. All the (actuator) sockets are connected to the ground layer. The parallel circuit has $i+1$ layers of circuit for $i - bit$ signals and one common ground. The model as shown in Figure 4-3 has two separated layers because the input signal is a 2-bit signal.

Another type of the circuit is a *serial circuit.* With the serial circuit, all of the parts are serially connected to the circuit. The layout of the serial circuit is generated by the sticker scaling algorithm; the algorithm is described in Figure 6-8 and Section 6.1.3. The serial circuit has i layers of circuit for $i - bit$ signals. One end of each layer is connected to ground; each layer has two ends: + and -.

The parallel circuit is good for the low voltage sticker controller while the serial circuit is good for the low ampere sticker controller. We used the serial circuits for our experiments described in Chapter 6.

4.1.2 The Signal Interface

The signal interface includes input ports for the signals and the power. By providing a sequence of the signals to the input port, we run the sticker controller to control the sheet.

The signal interface for this model has two input ports (Fig. 4-3). Each input port is connected to the corresponding input port of the sticker controller unit (by the circuit).

4.1.3 The Socket

We insert actuators on the (actuator) sockets. Each socket has o input ports, where o is the length of the actuator code of the actuator model (Sec. 3.1.1). The circuit connects each output port of the sticker controller units to each input port of the sockets. When a socket receives the actuator code, the socket passes the code to the actuator.

4.1.4 Sticker Controller Unit

Sticker places are locations within the controller substrate for the program (Fig. 4-4). A *sticker controller unit* is a group of sticker places for each 1 × 1 sticker controller. A 2 × 2 sticker controller has four sticker controller units. The unit has *sticker places*, *input ports*, and *output ports*. The input ports are connected to the circuit. The output ports are connected to three actuator sockets. When the input ports of the unit receive energy, the unit passes the signal to the selected outputs. We select the outputs by adding conductive material to the selected sticker places. Figure 4-5 shows a 3-1-1 sticker controller unit with no sticker.

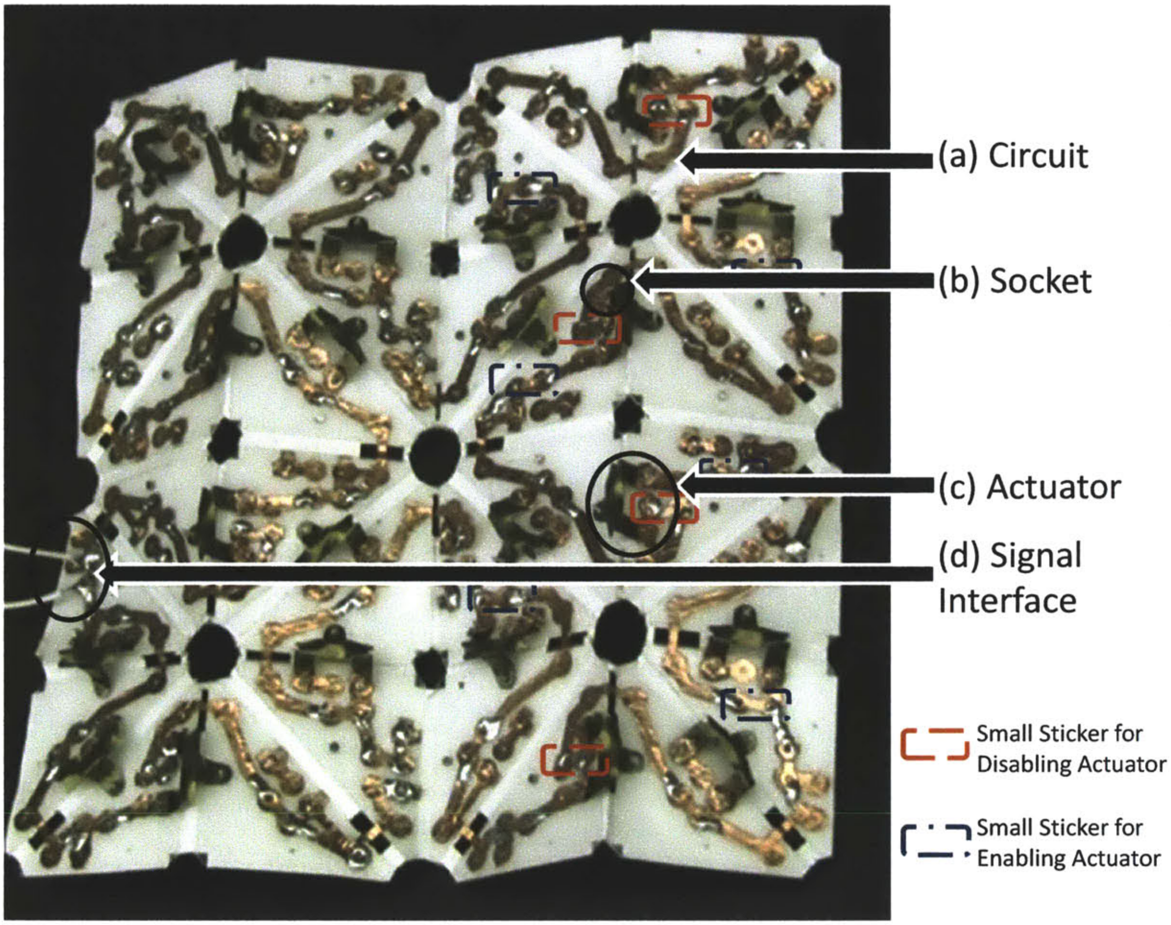

(a) Circuit
(b) Socket
(c) Actuator
(d) Signal Interface
Small Sticker for Disabling Actuator
Small Sticker for Enabling Actuator

Figure 4-4: 4 × 4 Self-folding sheet with executable sticker (Vertical folding program).

Figure 4-6 shows a 3-1-1 sticker controller unit with stickers. In Figure 4-6 (a), when input port I_a gets energy, O_{1a} and O_{2a} receive the energy; O_{1a} and O_{2a} are connected to S_{1a} and S_{2a}. This causes the actuators connected to O_{1a} and O_{2a} to be activated. The input voltage of I_a and the output voltage of O_{1a} and O_{2a} are the same.

We draw a model for the sticker controller unit in three different diagram (Fig. 4-5 and 4-6): a *model diagram* (a), a *code diagram* (b), and a *sticker diagram* (c). The model diagram shows detailed information of the sticker controller unit. The same information can be abstracted to a set of actuator codes displayed by the code diagram. It is also depicted in the sticker diagram as a graphical image.

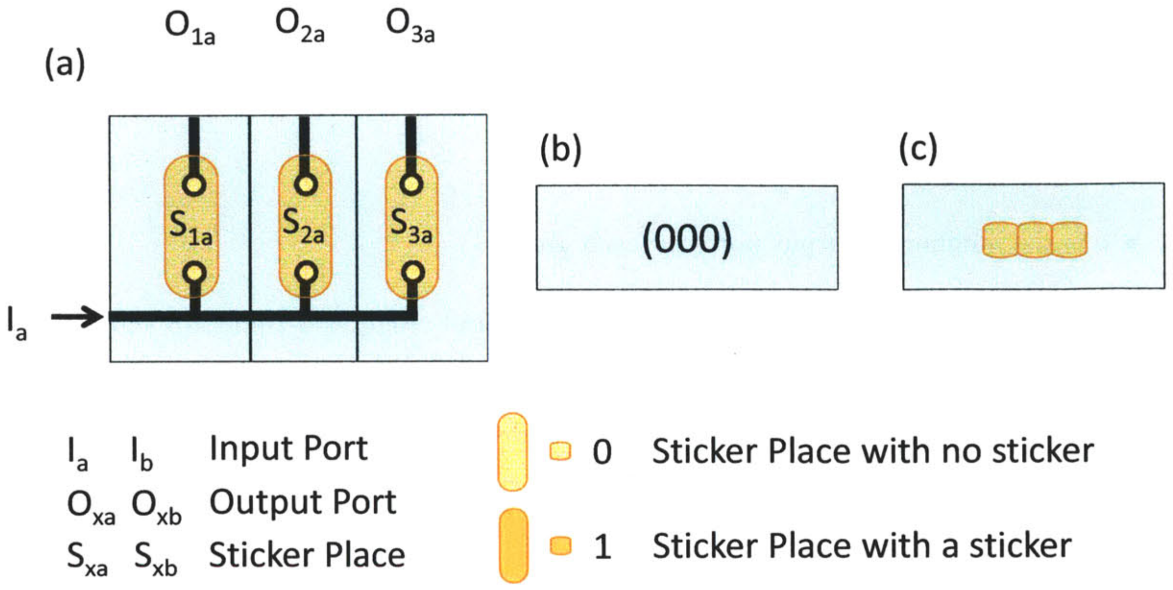


Figure 4-5: Three diagrams for the 3-1-1 sticker controller unit with no sticker. A small arrow mark on the left is the input port. (a), (b), and (c) represent the same sticker controller unit. (a) is the model diagram for the controller unit. (b) is the code diagram for the controller unit. (c) is the sticker diagram for the controller unit.

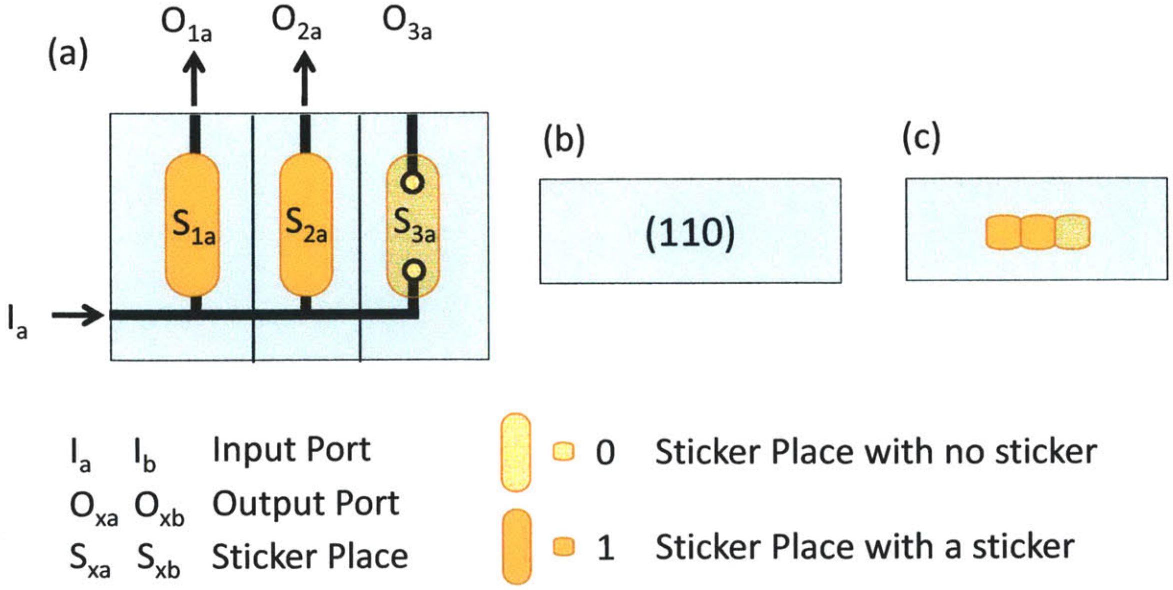


Figure 4-6: Three diagrams for the 3-1-1 sticker controller unit with the stickers. (a), (b), and (c) represent the same sticker controller unit. (a) is the model diagram for the controller unit. (b) is the code diagram for the controller unit. (c) is the sticker diagram for the controller unit.

A sticker controller unit is named by the *n-i-o* sticker controller unit, where:

- n is the number of groups of output ports,
- i is the number of input ports, and
- o is the number of output ports for each group.

Figures 4-5 and 4-6 show the 3-1-1 sticker controller unit. Figures 4-7 and 4-8 show a 3-3-1 sticker controller unit and a 3-3-2 sticker controller unit. All the sticker controller units are drawn in three different diagrams: model, code, and sticker diagrams (Fig. 4-5, 4-6, 4-7, and 4-8).

Each sticker controller unit has three groups of output ports. Usually one group of output ports is connected to one (actuator) socket. Each 1×1 self-folding sheet module has three actuators: left, diagonal, and bottom actuators, as shown in Figure 4-9. The first group (from left) of outputs is connected to the left (actuator) socket, the second group is connected to the diagonal socket, and the third group is connected to the bottom socket. Figure 4-10(left) shows a simplified model for the the 1×1 sticker controller.

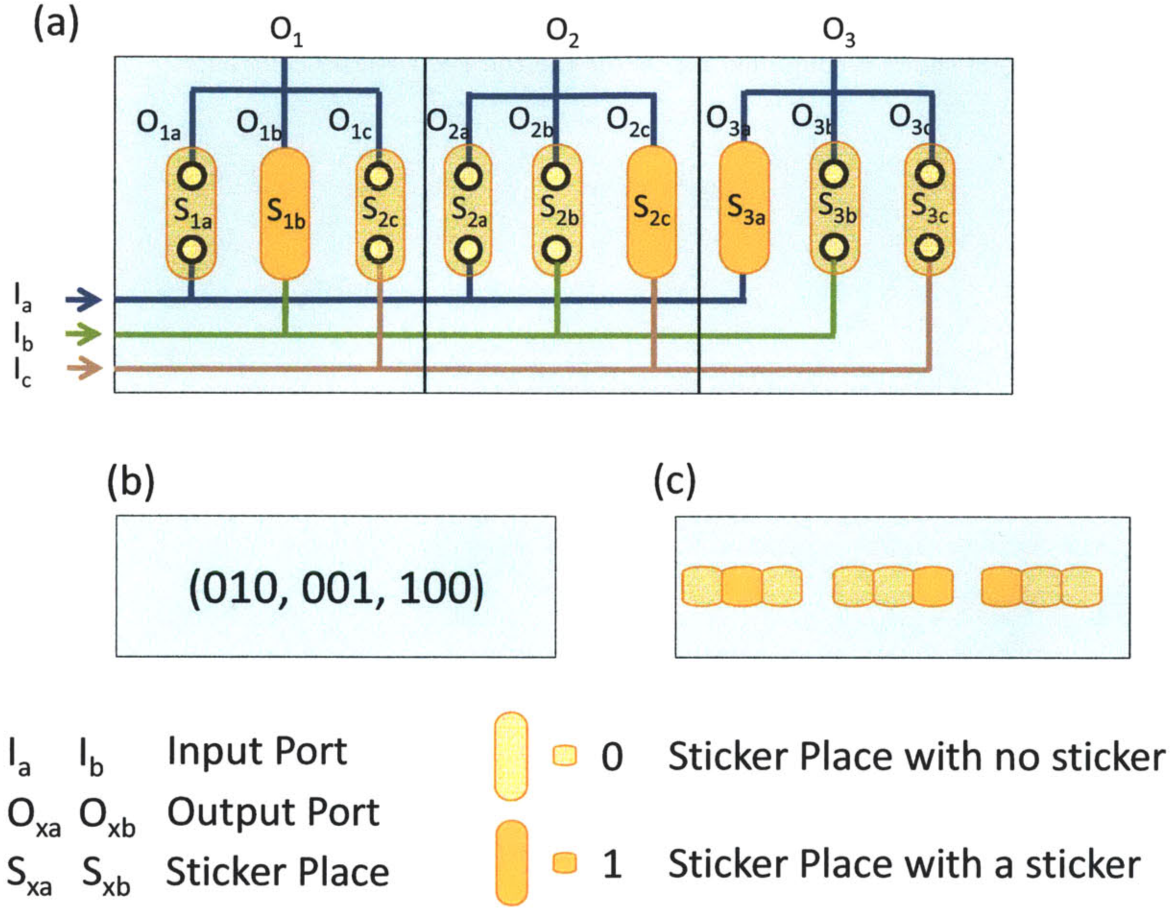


Figure 4-7: Three diagrams for the 3-3-1 sticker controller unit with stickers. (a), (b), and (c) represent the same sticker controller unit. (a) is the model diagram for the controller unit. (b) is the code diagram for the controller unit. (c) is the sticker diagram for the controller unit.

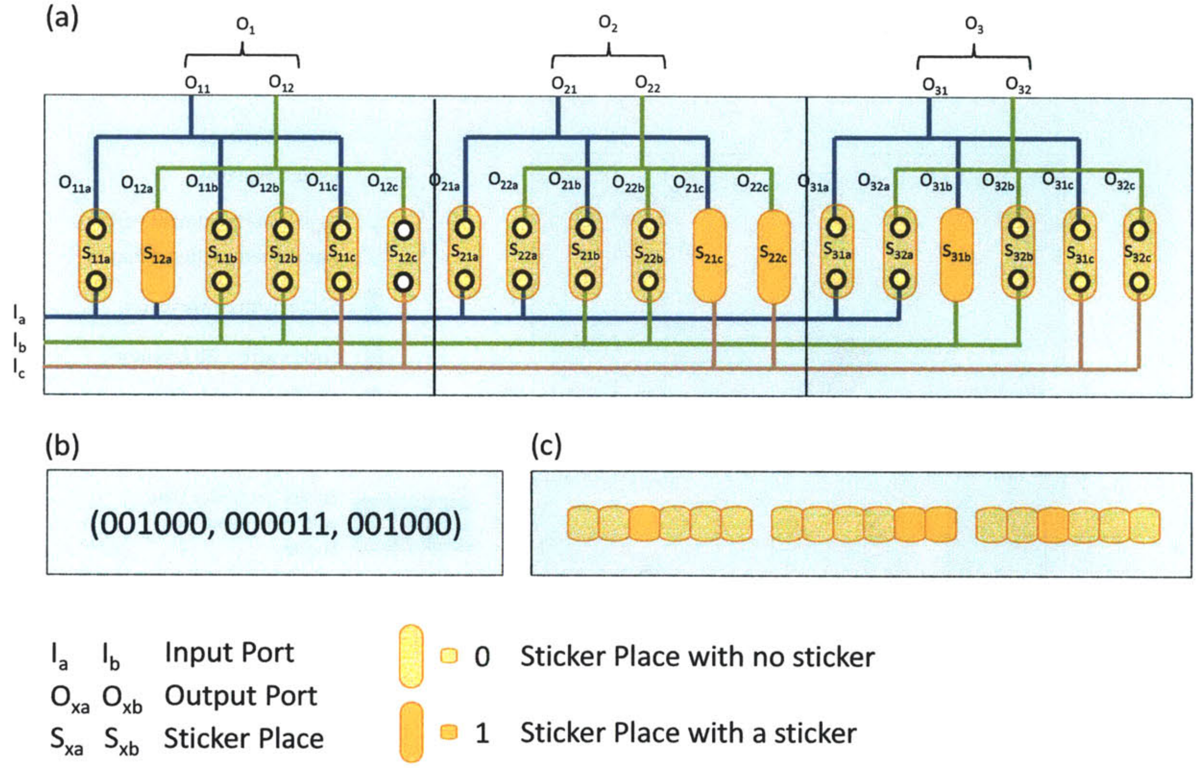

(a)
O1
O11 O12
O2
O21 O22
O3
O31 O32
O11a O12a O11b O12b O11c O12c
O21a O22a O21b O22b O21c O22c
O31a O32a O31b O32b O31c O32c
S11a S12a S11b S12b S11c S12c
S21a S22a S21b S22b S21c S22c
S31a S32a S31b S32b S31c S32c
Ia
Ib
Ic
(b)
(001000, 000011, 001000)
(c)
Ia Ib Input Port
Oxa Oxb Output Port
Sxa Sxb Sticker Place
0 Sticker Place with no sticker
1 Sticker Place with a sticker

Figure 4-8: Three diagrams for the 3-3-2 sticker controller unit with stickers. (a), (b), and (c) represent the same sticker controller unit. (a) is the model diagram for the controller unit. (b) is the code diagram for the controller unit. (c) is the sticker diagram for the controller unit.

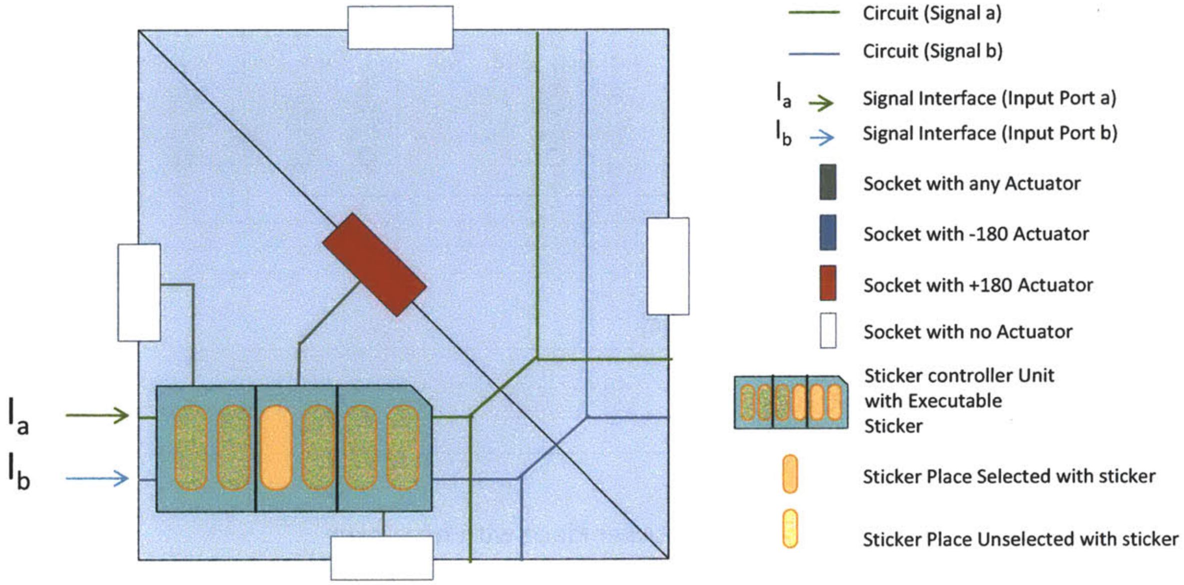

Circuit (Signal a)
Circuit (Signal b)
I_a Signal Interface (Input Port a)
I_b Signal Interface (Input Port b)
Socket with any Actuator
Socket with -180 Actuator
Socket with +180 Actuator
Socket with no Actuator
Sticker controller Unit with Executable Sticker
Sticker Place Selected with sticker
Sticker Place Unselected with sticker
I_a
I_b

Figure 4-9: Model for the 1 × 1 sticker controller. The socket is connected to ground; In model diagram, the wires for ground are eclipsed (Sec. 4.1.1).

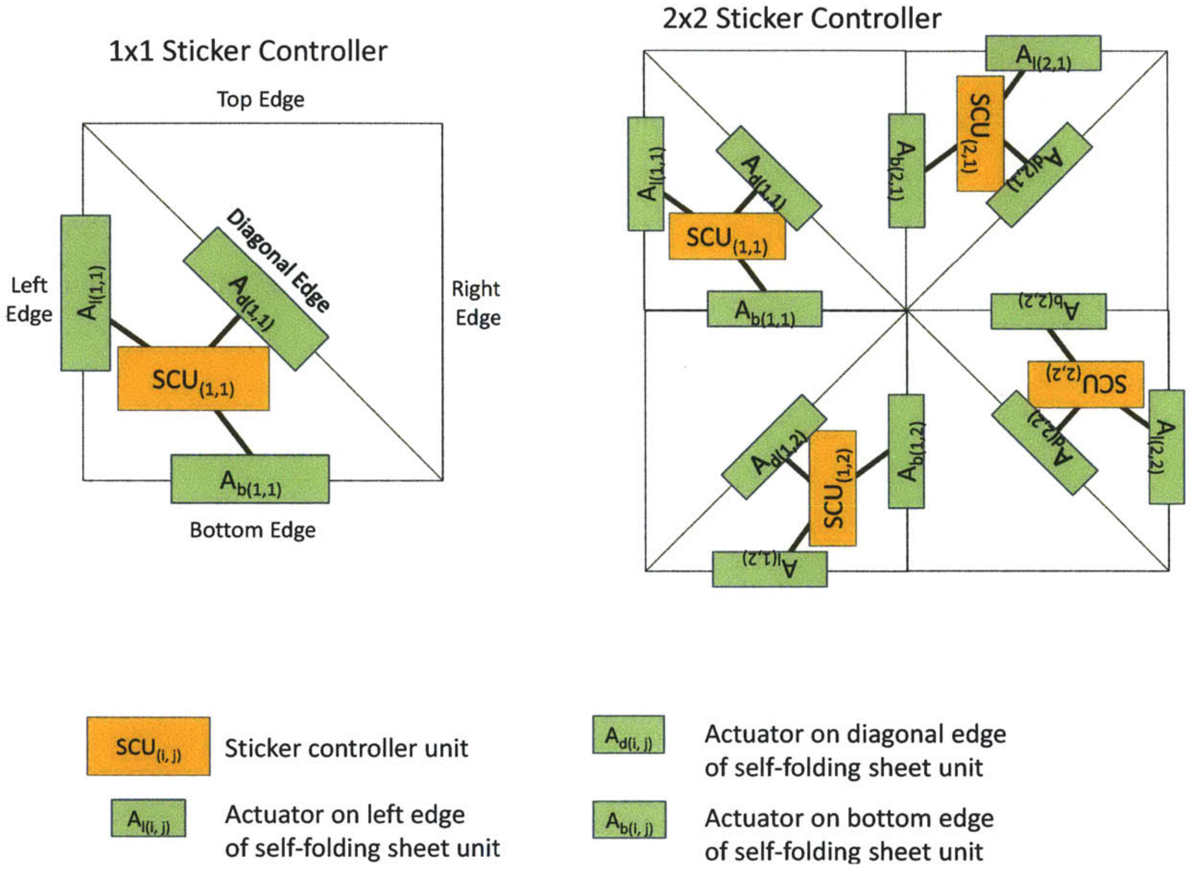


Figure 4-10: Simplified model for the 1×1 and 2×2 sticker controllers. A 2×2 self-folding sheet is composed of four 1×1 sticker controllers.

4.1.5 $m \times n$ Sticker Controller

A 1 × 1 sticker controller is the basic module of sticker control. An $m \times n$ sticker controller is composed of m columns and n rows of 1 × 1 sticker controllers.

The design of the sticker controller is modular. Figure 4-10 shows the simplified model for the 1 × 1 and 2 × 2 sticker controllers. The 2 × 2 sticker controller is composed of four 1 × 1 sticker controllers. The right edge of each 1 × 1 sticker controller is connected to the bottom edge of the neighboring 1 × 1 sticker controller.

The 4 × 4 sticker controller is also composed of four 2 × 2 sticker controllers (see Figure 4-11 that shows a 4 × 4 sticker controller). The right edge of the top-left 2 × 2 sticker controller is connected to the left edge of the top-right 2 × 2 sticker controller. The bottom edge of the top-left 2 × 2 sticker controller is connected to the top edge of the bottom-left 2 × 2 sticker controller.

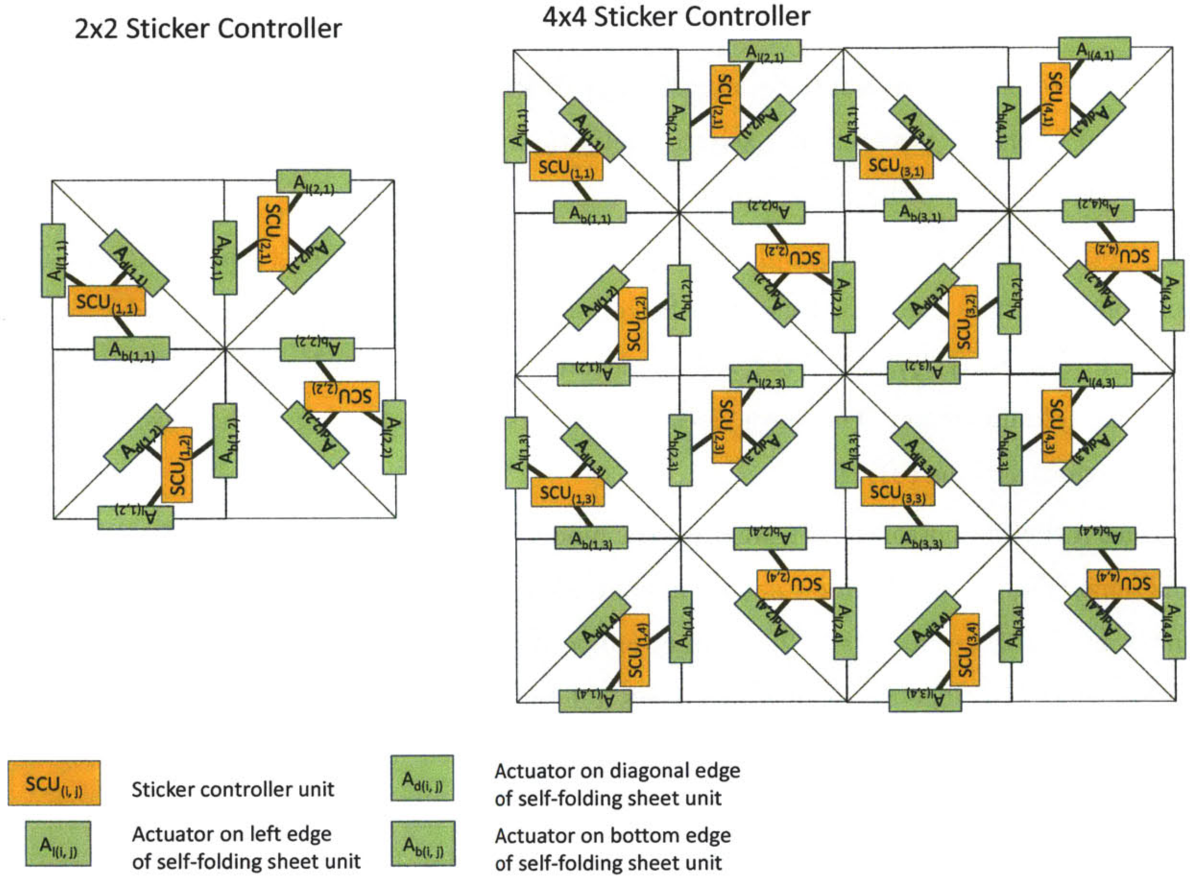


Figure 4-11: Simplified model for the 2×2 and 4×4 sticker controllers. A 4×4 self-folding sheet is composed of four 2×2 sticker controllers.

4.1.6 Sticker Controller with Executable Sticker

The executable sticker is a set of patches that are added by the user to the selected sticker places, according to the desired set of target shapes for the self-folding sheet. The executable sticker works like analogous to the memory of a computer that has the program and data. The executable sticker is also called a sticker program.

The sticker pattern of the sheet contains information for how to fold the self-folding sheet for the target shapes. When the sticker fills the sticker places, the actuating signals, received by signal interface, passes to the selected actuators. By folding the selected actuators, the sheet transforms into the target shape.

Figure 4-12 shows an example of a sticker controller with an executable sticker. The controller controls the actuators for a target shape. When we input a signal (10) to the signal interface (I_1 I_2), the 1 × 1 3-2-1 controller activates the selected actuators for the diagonal folding shape. The sheet does not transform into any shape when we input any other signals such as (01), (11).

Like a sticker controller unit, we draw the model for the sticker controller in four different diagrams (Fig. 4-12): a target shape diagram(a), a model diagram(b), a code diagram(c), and a sticker diagram(d)(e). The target shape diagram shows the target shapes for the sticker controller. The model diagram shows a detailed model for the sticker controller. The information about the unit with stickers can be translated into codes. The code diagram shows the codes. The information about the unit with stickers can also be translated into a graphical image. The sticker diagram shows the graphical image. The locations of the sticker places are not fixed in the sticker diagram(Fig. 4-12 (d) and (e)).

Given various executable stickers, a sticker controller can select the actuators needed to transform the sheet into various shapes. Figures 4-13, 4-14, and 4-15 show examples of sticker controllers with various executable stickers.

Figure 4-13 shows another example of control for a target shape. The controller in this example is a 2 × 2 3-2-1 sticker controller. When we input the signal (10) to the signal interface (I_1 I_2), the 2 × 2 3-2-1 controller activates the selected actuators for diagonal

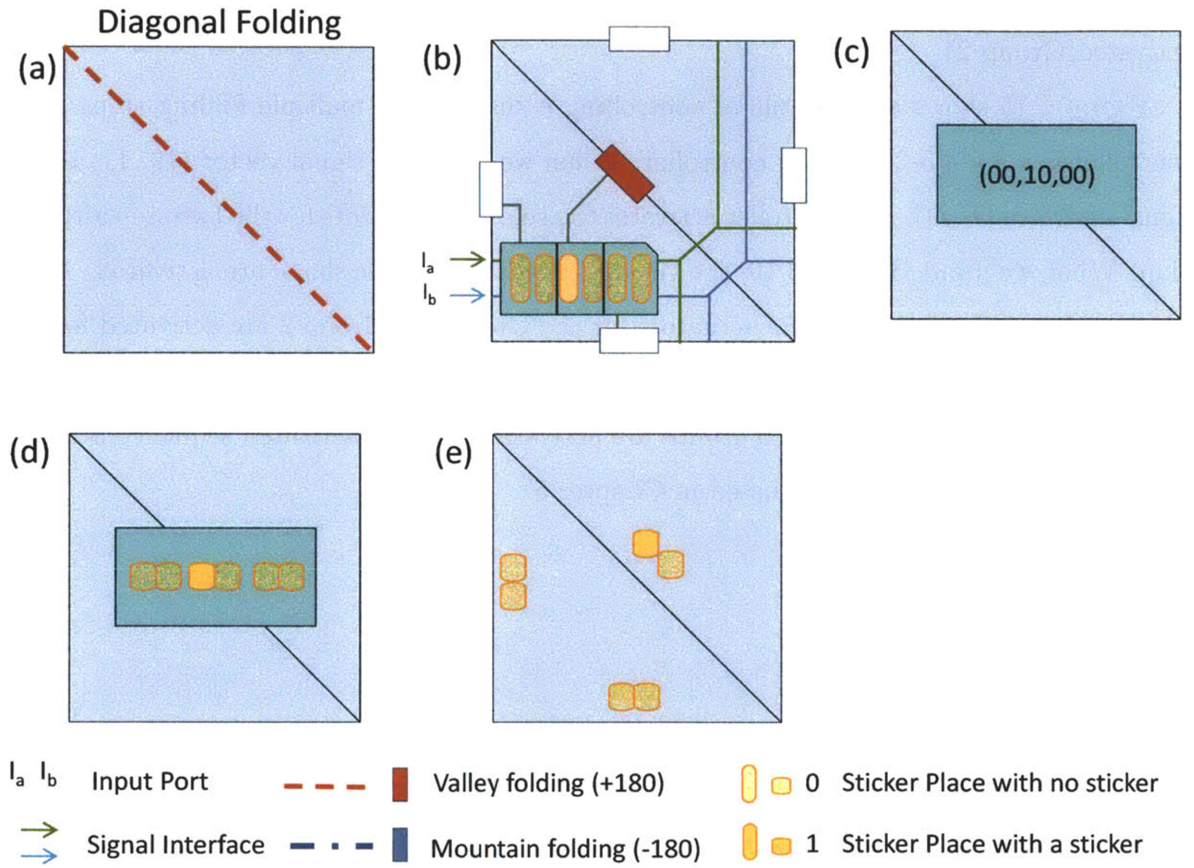


Figure 4-12: Example of 1 × 1 3-2-1 sticker controller with the executable sticker for a diagonal folding. (a) is the target shape diagram (diagonal folding). (b)(c)(d) and (e) show the same sticker controller with the same executable sticker. When the signal (10) is input, the controller transforms the sheet into the target shape (a). (b) is the model diagram. (c) is the code diagram. (d) and (e) are the sticker diagrams.

folding shape.

Figure 4-14 shows an example of a sticker controller controlling multiple shapes. The controller in this example is a 2×2 3-2-1 sticker controller. When we input the signal (10) to the signal interface (I_1 I_2), the actuators for the diagonal folding shape are activated(Group 1). When we input the other signal (01), the actuators for the vertical folding shape are activated(Group 2).

Figure 4-15 shows an example of controlling a shape using multiple folding steps. The controller is a 4×4 3-2-1 sticker controller. When we input the signal vector (10, 11) to the signal interface (I_1 I_2), the controller activates the selected actuators for the two step airplane plan. When we input the signal 10, the Group 1 actuators for the shape are activated. Next, when we input the signal 11, the actuators for the Group 1 $\cup$ Group 2 are activated for the two step airplane shape.

The process of selecting which groups are activated, and the activated sequence is called sticker programming and is discussed in Chapter 5.

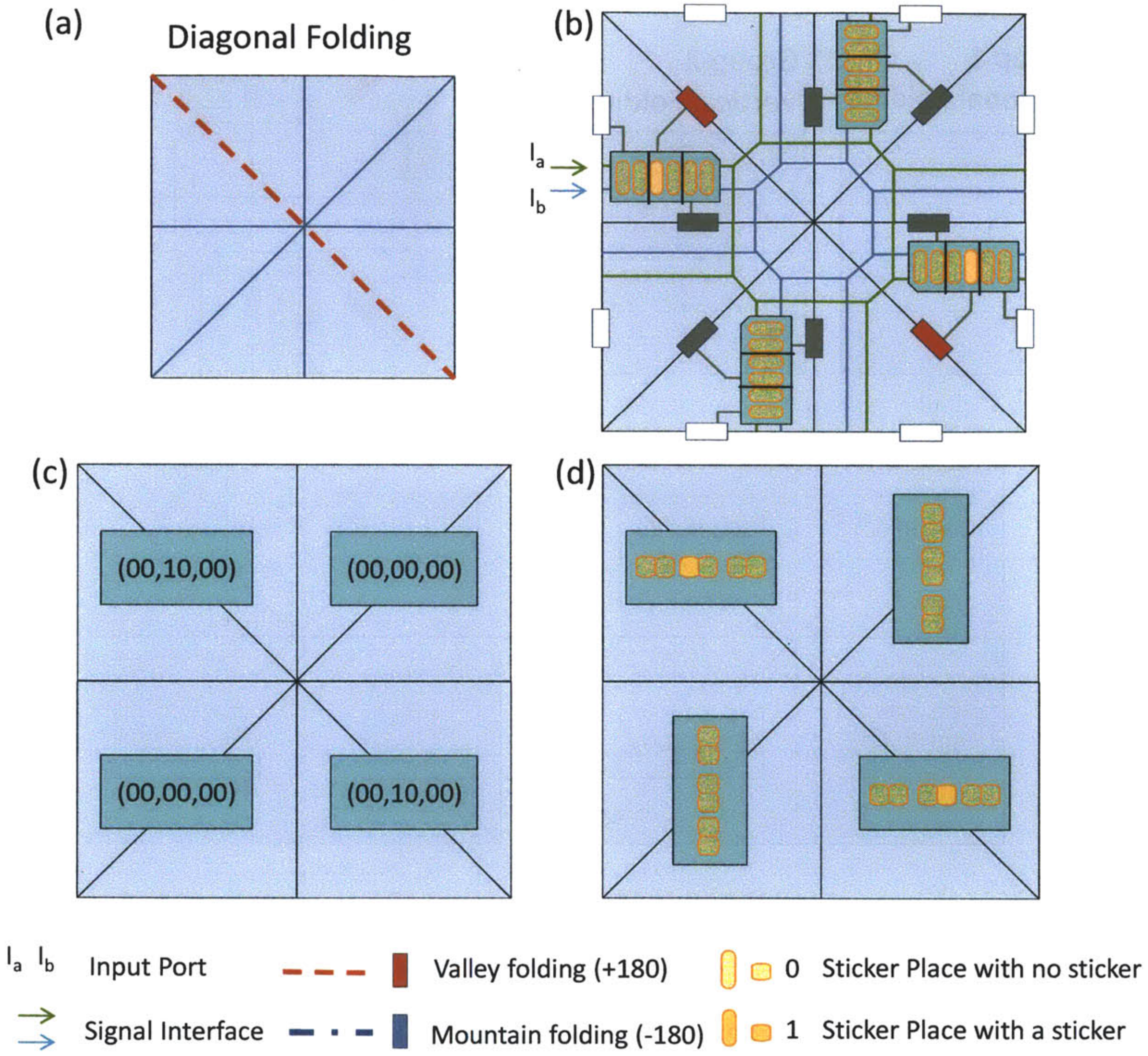


Figure 4-13: Example of 2 × 2 3-2-1 sticker controller with executable stickers for diagonal folding. (a) is the target shape diagram (diagonal folding). (b)(c) and (d) show the same sticker controller with the same executable sticker. When the signal (10) is input, the controller transforms the sheet into the target shape (a). (b) is the model diagram. (c) is the code diagram. (d) is the sticker diagram.

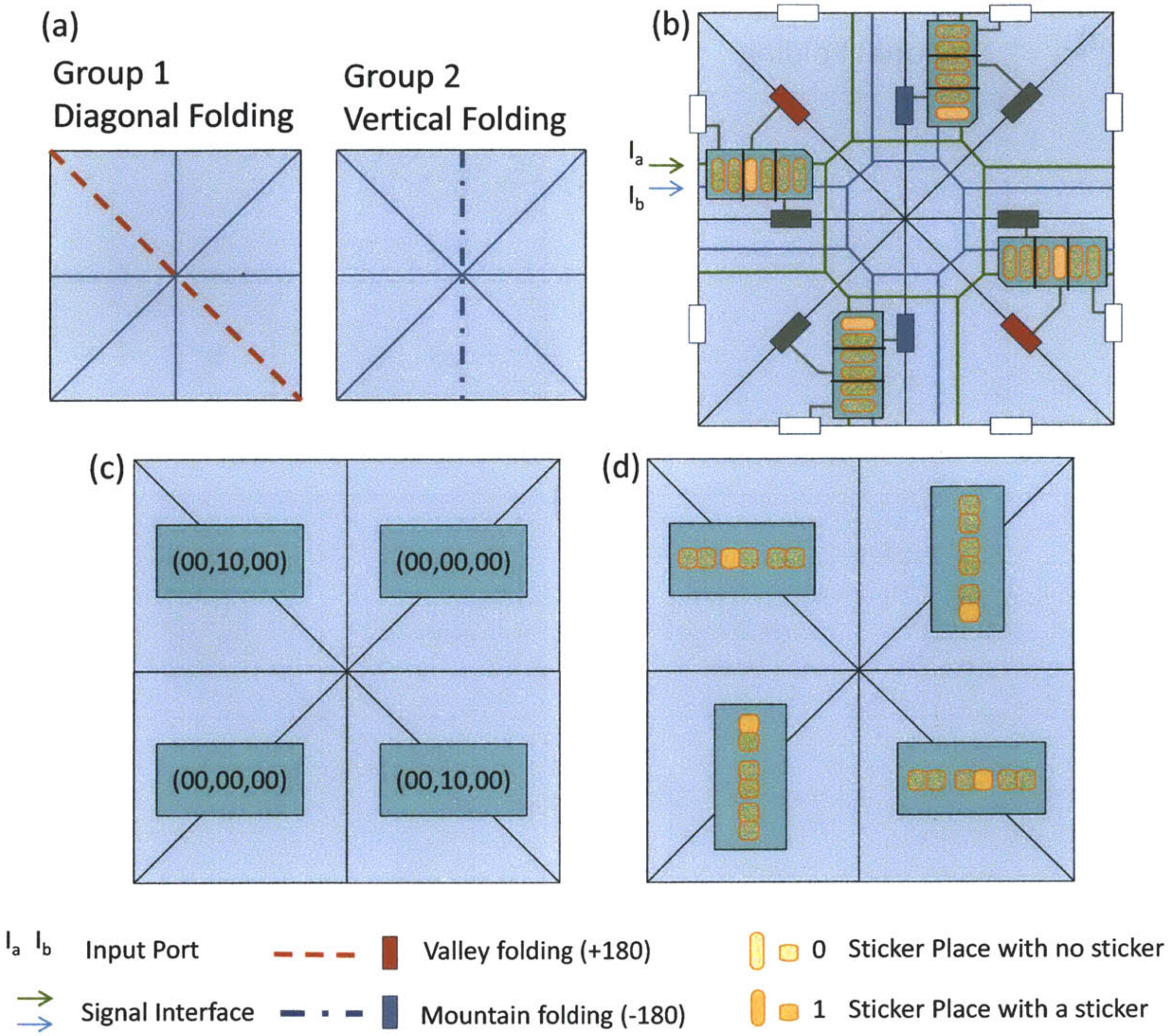


Figure 4-14: Example of 2 × 2 3-2-1 sticker controller with the executable sticker for multiple shape folding: (1) diagonal folding and (2) vertical folding. (b)(c) and (d) show the same sticker controller with the same executable sticker. When the signal (10) is input, the controller transforms the sheet into Group 1 of the target shape ((a)left). When the signal (01) is input, the controller transforms the sheet into Group 2 of the target shape ((a)right). (a) is the target shape diagram (diagonal folding and vertical folding). (b) is the model diagram. (c) is the code diagram. (d) is the sticker diagrams.

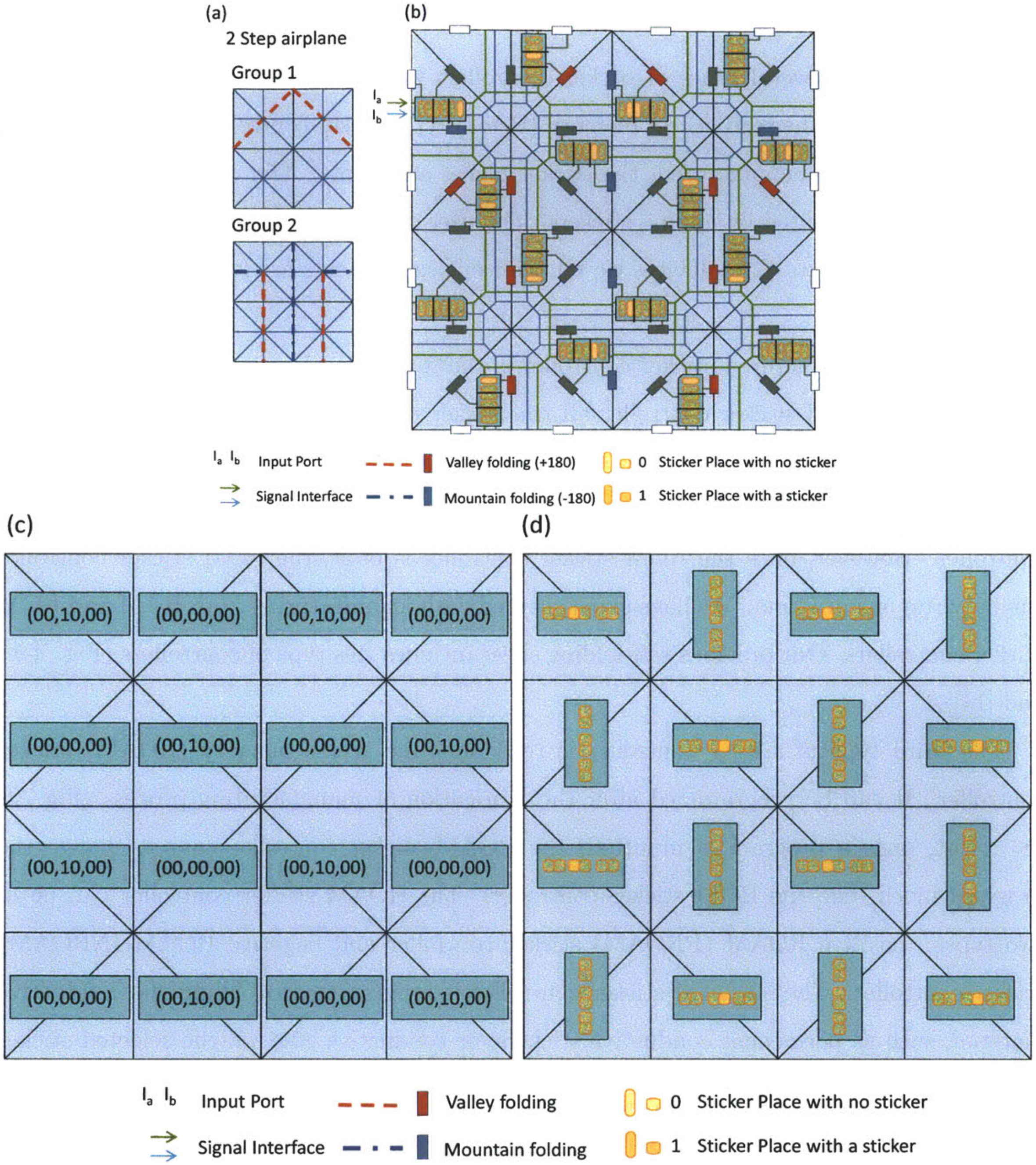


Figure 4-15: Example of 4 × 4 3-2-1 sticker controller with the executable sticker for the two step airplane shape. (a) is the target shape diagram. (b)(c) and (d) show the same sticker controller with the same executable sticker. When signals 10 and 11 are input sequentially, the controller transforms the sheet into the target shape (a). (b) is the model diagram. (c) is the code diagram. (d) is the sticker diagram.

4.2 Types of the Sticker Controllers

We have designed several types of sticker controllers with different capabilities. The basic sticker controller is the read access memory (RAM) type sticker controller. When the RAM sticker controller is manufactured, the sticker places are empty. The user's desired control sequence is implemented by adding stickers. The user can change the program by replacing the executable stickers. The devices we built for our experiments are RAM type (Fig. 4-16, and Ch. 6).

The other manufactured sticker controller is the read only memory (ROM) sticker controller. In the ROM sticker controller all the sticker places are filled with the conductors during manufacturing. Programming is achieved by eliminating the stickers to deactivate actuators. The ROM sticker controller is simpler to manufacture than the RAM sticker controller. However, once the ROM sticker controller is built, the ROM sticker controller can only run one program. We have experimented with manufacturing both RAM and ROM sticker controllers. Our previous self-folding sheet includes this type of controllers (Fig. 4-17 and [14]).

The third type of sticker controller is read one-time access memory (ROAM) sticker controller. It can be programmed after the fabrication or manufacturing process (Fig. 4-18). But, once a program is inputted, the ROAM sticker controller cannot change the program much, like the ROM sticker controller. The ROAM sticker controller can be of two types: positive ROAM (PROAM) sticker controller and negative ROAM (NROAM) sticker controller. The PROAM sticker controller is programmed by filling the conductive material, such as permenent conductive stickers, or conductive glue, on the selected sticker places. The NROAM sticker controller is programmed by disconnecting the circuit on the unselected sticker places. The laser cutter, heat, chemical reaction, or punch can be used for disconnection. Different fabrication processes are used for these various types of the sticker controllers. However, they all have the architecture described in Figure 4-19. Table 4.1 summarizes the differences.

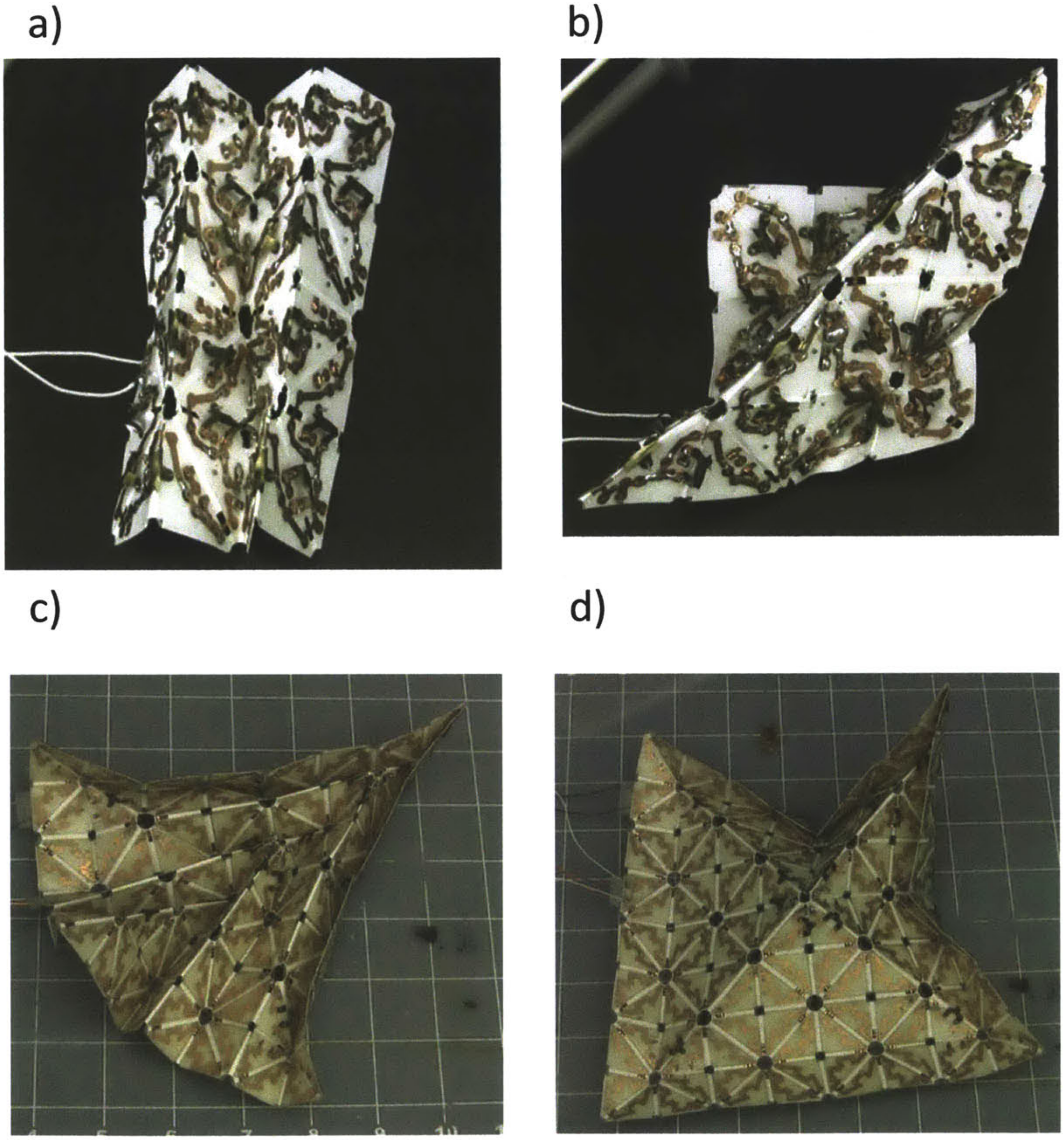


Figure 4-16: Self-Folding Sheets. (a) Vertical Folding (b) Diagonal Folding (c) Space Shuttle (d) Hat. The objects in (a) and (b) were made from a 4 × 4 sheet. The objects in (c) and (d) were made from an 8 × 8 sheet. (a)(b) 4 × 4 self-folding sheet. (c)(d) 8 × 8 self-folding sheet.

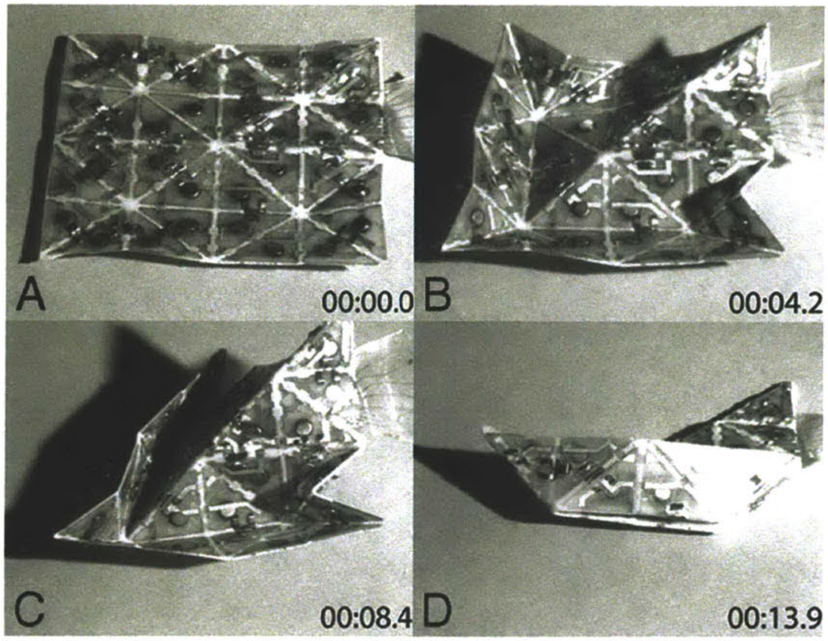


Figure 4-17: Example of read only memory (ROM) sticker controller for a boat [14]. Because the embedded controller is manually designed circuit for two shape (a boat and an airplane), we cannot change the program.

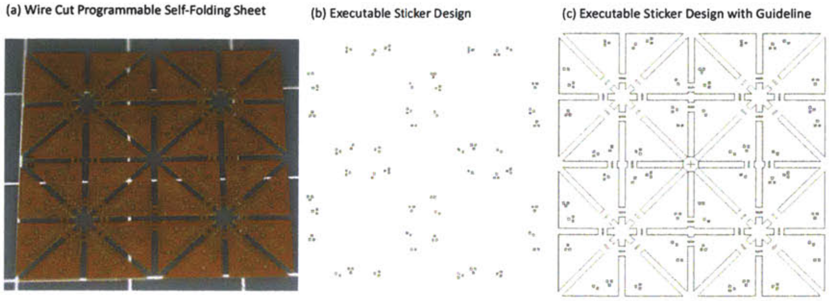

(a) Wire Cut Programmable Self-Folding Sheet
(b) Executable Sticker Design
(c) Executable Sticker Design with Guideline

Figure 4-18: Example of negative read one-time access memory (NROAM) controller. (a) 4 × 4 wire cut programmable self-folding sheet (W-sheet) with no program. (b) Executable sticker design having two shapes (a space-shuttle and a pyramid). (c) Executable sticker design with guide line. The small squares are Executable sticker design. The guide line is outline of the W-sheet. (a) W-sheet embeds a specially designed circuit. We program the W-sheet by making small square-shape holes according to (b).

Sticker Computer Type		Can Be Reprogrammed	Method of Programming
RAM		Yes	Sticker
ROM		No	Embedded Sticker or Program Embedded Circuit
ROAM	(PROAM)	Once	Permanent Sticker or Conductive Glue
	(NROAM)	Once	Removing Conductive Material

Table 4.1: Characteristics of Sticker Computers

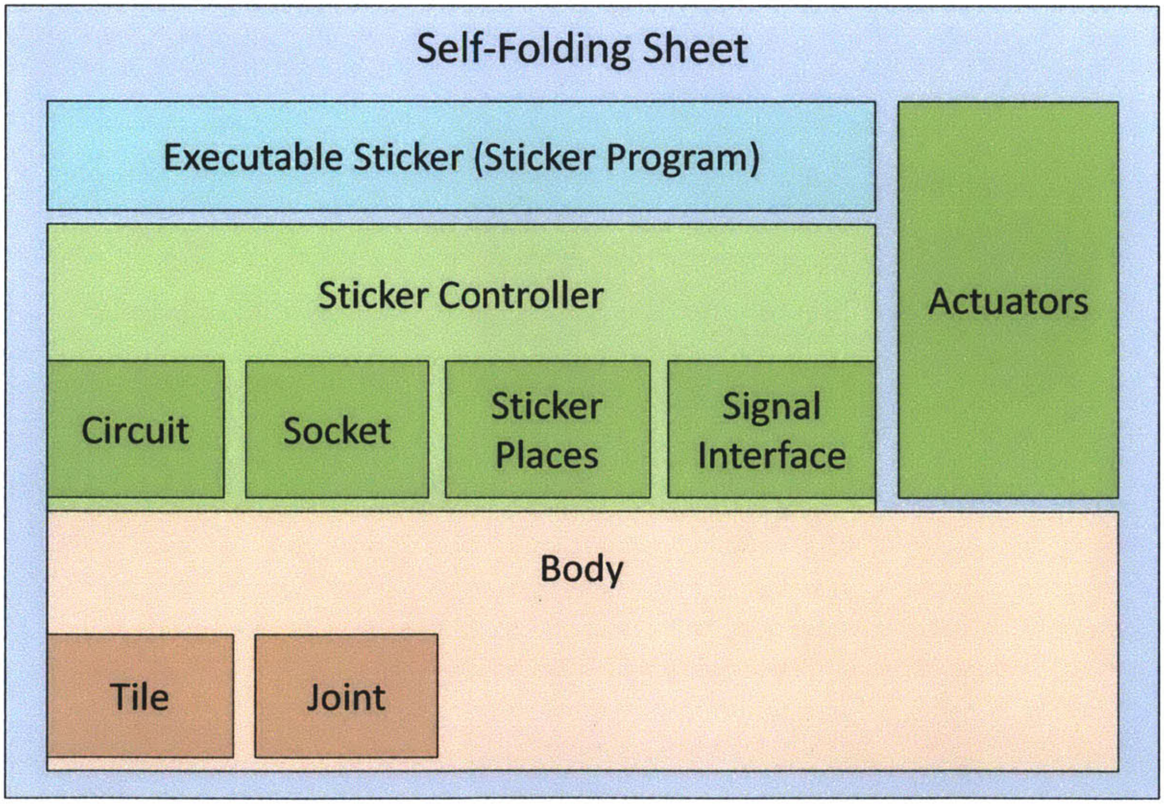


Figure 4-19: Components of Sticker Controller Architecture

Chapter 5

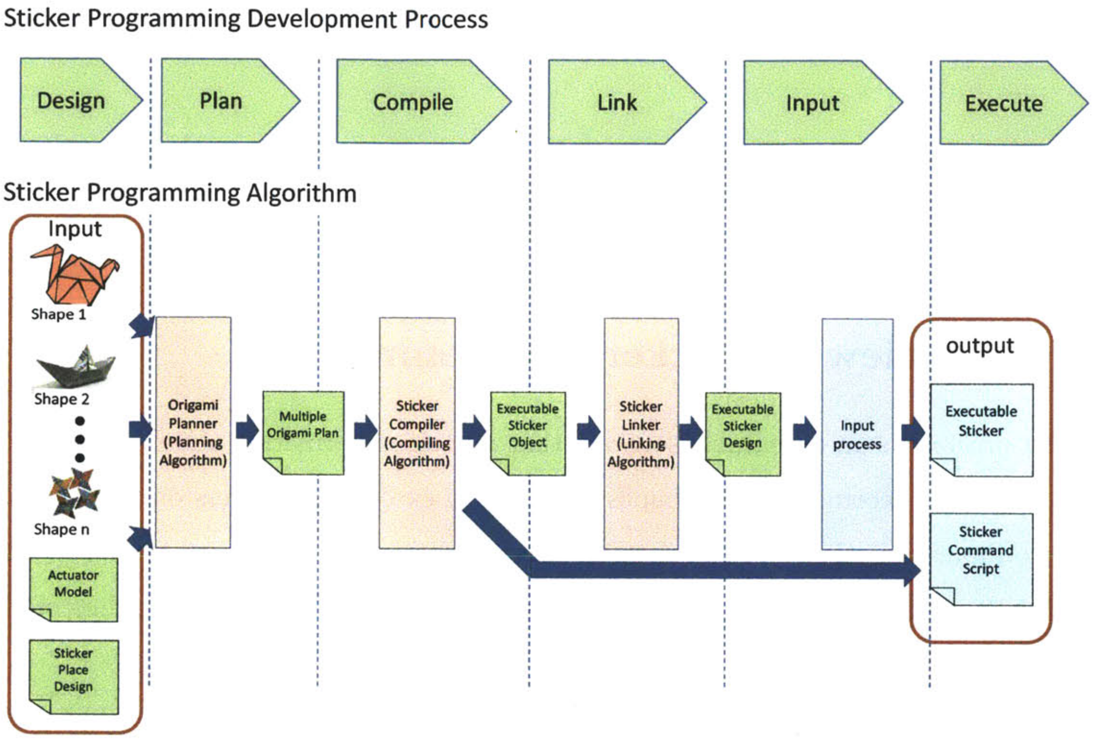
Sticker Programming

5.1 Overview of Sticker Programming

A *sticker program* is an executable software that contains information about how edges of a self-folding sheet become origami shapes. By placing executable stickers on the self-folding sheet, we input the sticker program to the sticker controller. When we send the signals, according to sticker command script, the sticker controller controls the self-folding sheet for the origami shapes.

In this chapter, we present and analyze a new programming method, which we call *sticker programming*, for designing, implementing, and executing sticker programs. Figure 5-1 shows the stages of the sticker programming development process and the algorithms used in each stage. With this process, users can easily develop sticker programs containing multiple origami shapes and execute the programs on self-folding sheets.

The sticker programming algorithm supports the sticker programming by automatically generating executable sticker designs and executable command scripts. In a previous work, we presented origami planning algorithms that automatically generates origami plans[2] from origami shapes. The sticker programming algorithm has similar flavor and automatically compiles the plans into executable sticker designs and executable command scripts.



Sticker Programming Development Process

Design → Plan → Compile → Link → Input → Execute

Sticker Programming Algorithm

Input
Shape 1
Shape 2
Shape n
Actuator Model
Sticker Place Design
Origami Planner (Planning Algorithm)
Multiple Origami Plan
Sticker Compiler (Compiling Algorithm)
Executable Sticker Object
Sticker Linker (Linking Algorithm)
Executable Sticker Design
Input process
output
Executable Sticker
Sticker Command Script

Figure 5-1: Overview of sticker programming. The diagram shows stages of sticker programming development process (top) and processes of the sticker programming algorithm for each stage (bottom). (Design Stage) The user selects the design. (Plan Stage) The planner computes the folding sequence and the placement of actuators. (Compile Stage) The compiler computes the machine codes for the folding sequence and the placement of stickers. (Link Stage) The linker computes the design of the stickers. (Input Stage) The user places the stickers to the self-folding sheet. (Execute Stage) Finally, the self-folding sheet is finalized for the desired object and the folding sequence is triggered by applying voltage.

5.2 Sticker Programming Development Process

The sticker programming development process gives the outline for generating sticker programs that we explain the six stages of the process: design, plan, compile, link, input, and execute (Fig. 5-1).

5.2.1 Design

The objective here is to design target shapes and to select an appropriate self-folding sheet. To build a sticker program, the target shapes, and a *sticker place design* and an actuator model (Ch. 3) are required.

A sticker place design and actuator model are determined by selecting the self-folding sheet. Details of the sticker place design and actuator model will be explained in Section 5.3.1.

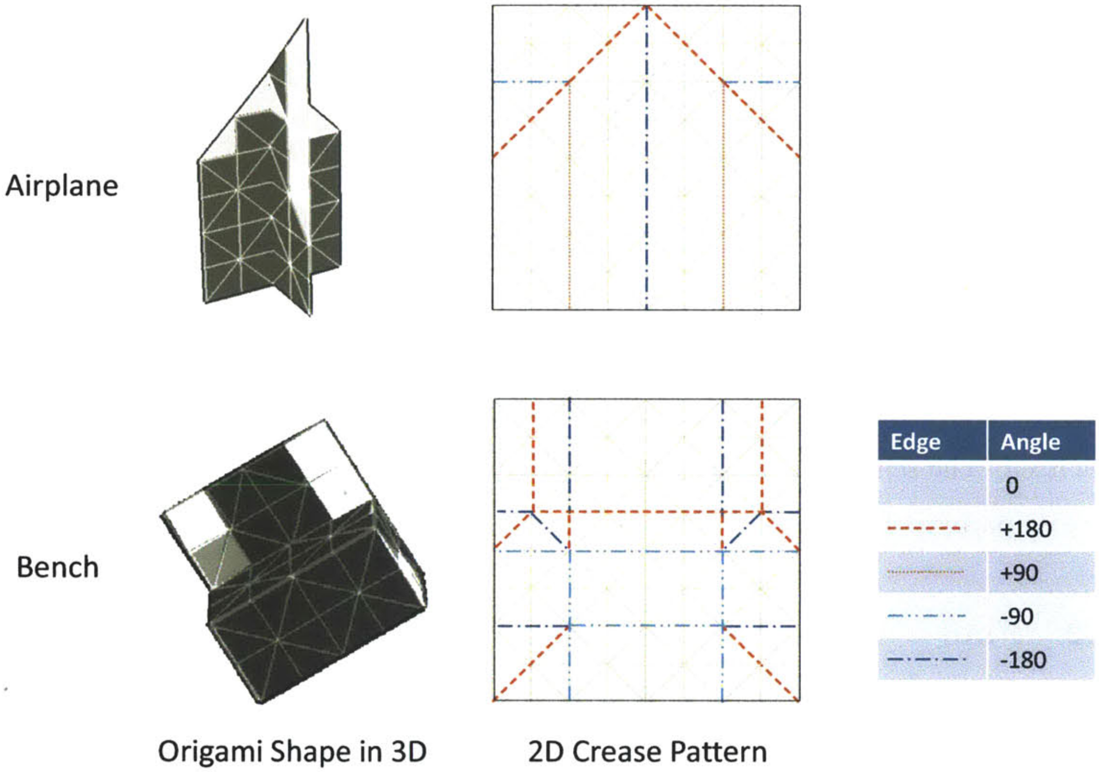
Target shapes are box-pleated 3D origami shapes into which the self-folding sheet transforms itself. The self-folding sheet is 2D thin material. 3D target shape can be drawn as a 2D crease pattern. Figure 5-2 shows two target shapes.

5.2.2 Plan

The objective is to build a *multiple origami plan* from k target shapes. The origami plan contains folding sequences directing how the self-folding sheet should transform itself into the target shapes.

First, we use two algorithms: a *single origami planner* and a *multiple origami planner* described in detail in [2]. Given k origami shapes, we apply the single origami planner to each input target shape individually, producing a single origami plan for folding each single shape. Then, we use multiple origami planner. When we input the single origami plans into the multiple origami planner, the planner automatically optimizes and generates a origami plan[1]. We implemented these planners (these planning algorithms) with Java [2]. In [2], we

[1] The multi-origami plan merges the individual plans so that each of the objects can be folded by the aggregate plan. It further optimizes the placement of the actuators needed to execute the plan.



Edge	Angle
	0
- - - - - -	+180
···········	+90
-·-·-·-	-90
—·—·—	-180

Figure 5-2: Target shapes of airplane and bench drawn in 2D crease pattern and the crease patters (with mountain and valley folds) for actuating the shapes. A multi-origami sheet will include both of these crease patterns.

describe in great detail that single origami planner and the multiple origami planner.

5.2.3 Compile

The objective is to generate an *executable sticker object* and a sticker command script from an origami plan, using a *sticker compiler.* The sticker compiler is an algorithm converting a origami plan to an executable sticker object and sticker command script. The details of the compiler are in Section 5.3.2.

The sticker command script contains sequences of commands for the target shapes. According to the script, we send signals to a sticker controller (Ch. 4). Following the commands, the sticker controller executes the executable sticker and controls the self-folding sheet.

Like the executable sticker object, the sticker command script is also used for any self-folding sheet that meets the specifications of our architecture.

5.2.4 Link

The objective of the link stage is to generate an executable sticker design containing graphical data (like a CAD file).

This stage is supported by the *sticker linker* (Fig. 5-1) Given the executable sticker object and the sticker place design, the sticker linker generates the executable sticker design. The details of the sticker linker are described in Section 5.3.2.

5.2.5 Input

In the input stage, we provide a physical executable sticker to the sticker controller. Although there is only one sticker controller architecture, there are different implementations of sticker controllers. Each implementation has its own input process of the executable sticker. The input process used for our experimentation will be described in Chapter 6.

5.2.6 Execute

In this stage, the user executes the sticker program on the sticker controller and transforms the self-folding sheet into target shapes on-demand. We input signals, according to a sticker command script. Then, the controller controls the edges of the sheet to transform into the target shapes.

5.3 Sticker Programming Algorithm

Given a self-folding sheet and target shapes, the sticker programming algorithm generates an executable sticker design and a sticker command script. A fixed self-folding sheet contains its actuator model and its sticker place design. We input an executable sticker to the self-folding sheet according to the executable sticker design and run the self-folding sheet according to the sticker command script.

The sticker programming algorithm is composed of three components: an origami planner, a sticker compiler, and a sticker linker (Fig. 5-1).

Given multiple target shapes, the origami planner generates the origami plan (Fig. 5-3).

Given an actuator model of a self-folding sheet and an origami plan, the sicker compiler generates an executable sticker object and an executable command script; the sticker object is a machine code containing group information of the origami plan and the sticker script is a machine code containing shape information of the origami plan.

Given a sticker place design of a self-folding sheet and an executable sticker object, the sticker linker generates an executable sticker design.

Details about the origami planer are presented in [2]. In this section, we focus on the sticker compiler and the sticker linker.

5.3.1 Problem Formulation

In this section, we define the terminology we need for the sticker programming algorithm. Details of self-folding sheet models, actuator models.

Origami Plan

An origami plan (a multiple origami plan in [2]) is a folding plan with directions for how a self-folding sheet transforms itself into multiple origami shapes. Given desired shapes, the origami planer generates an optimized origami plan.

Definition 5.3.1 *An origami plan P = (G, S), where G is group information, and S is shape information.* $G = \{ g_1, g_2, ..., g_n \}$*, where an actuator group* $g_i = (V_i, E_i)$*, for* $1 \leq i \leq n$*.* $V_i = \{v_1, v_2, ..., v_m\}$ *is a set of vertices and* $E_i = \{e_1, e_2, ..., e_m\}$ *is a set of edges with angles. A edge* $e_j = (v_s, v_t, a_i)$*, where* $1 \leq j \leq n_e$ *(*n_e *is # of edges) and* $-180° \leq a \leq +180°$ *is a folding angle.*

S is a set of folding sequences $\{ o_1, o_2, ... o_n \}$*, where* $o_i = (s_1, s_2, ..., s_t)$ *is a folding sequence for an* i^{th} *origami shape and* $1 \leq i \leq n$*.* $s_j = \{ g \mid g \in G \text{ for } j^{th} \text{ step of folding for } o_i \}$*.*

Figure 5-3 shows an example of an origami plan for an airplane and a boat. Each shape is folded in one time step. The group information (top-left box) has three actuator groups while the shape information (bottom-left box) has two shapes. To transform the self-folding sheet into Shape 1, according to the shape information, the self-folding sheet folds all edges in Group 1 and Group 3 simultaneously (Group 1 ∪ Group 3 is the same as Shape 1, top-right). To transform the self-folding sheet into Shape 2, according to the shape information, the self-folding sheet folds all edges in Group 2 and Group 3 simultaneously (Group 2 ∪ Group 3 is the same as Shape 2, bottom-right).

Figure 5-4 is another example of an origami plan for a bench and a boat. The transformation of the bench (Shape 1) requires two time steps (as shown in Shape 1 (top-left)). In the shape information, Shape 1 has a vector containing two sets of the groups sequentially ({ Group 1 }, { Group 2, Group 3 }). When the self-folding sheet transforms itself into the bench, it folds the edges in Group 1. Then, it simultaneously folds the edges in Group 2 and Group 3 (Group 1 is the time step t_1 of Shape 1; Group 2 ∪ Group 3 is the same as the time step t_2 of Shape 1, top-right; Group 3 ∪ Group 4 is the same as Shape 2, bottom-right.

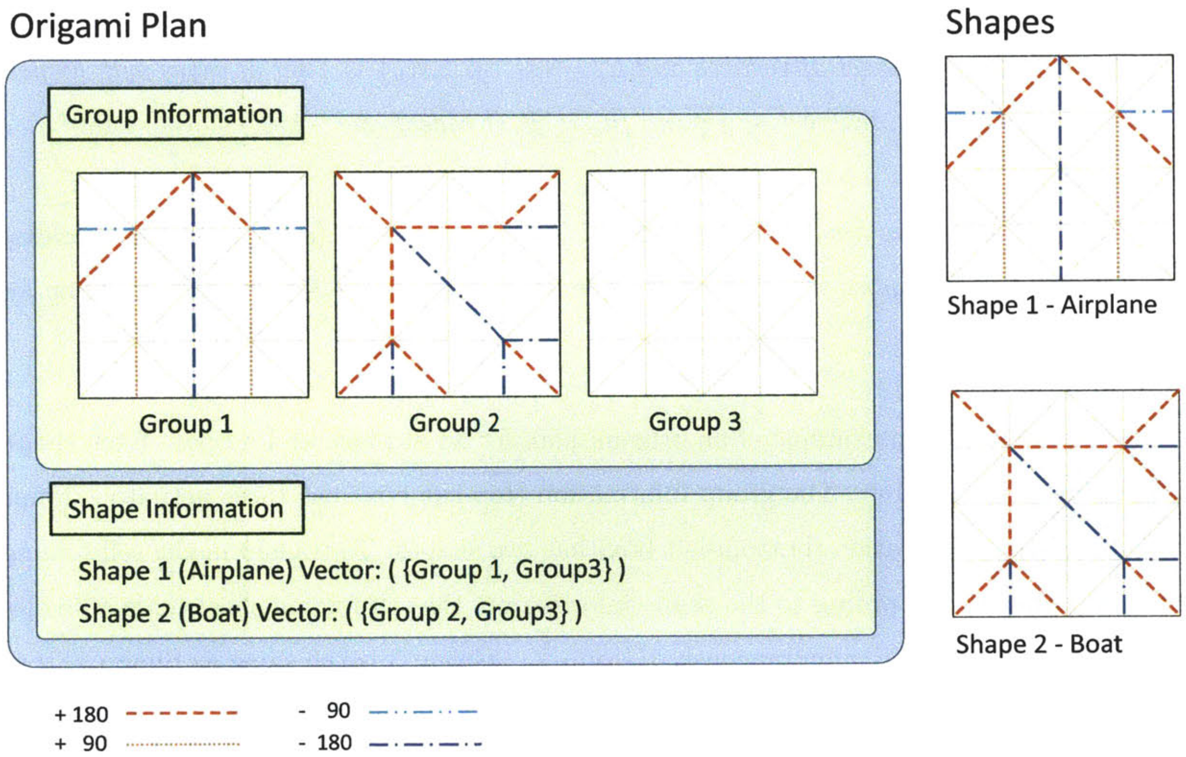


Figure 5-3: Origami plan for airplane and boat. All edges in group i are folded in parallel. Different groups are controlled in sequence.

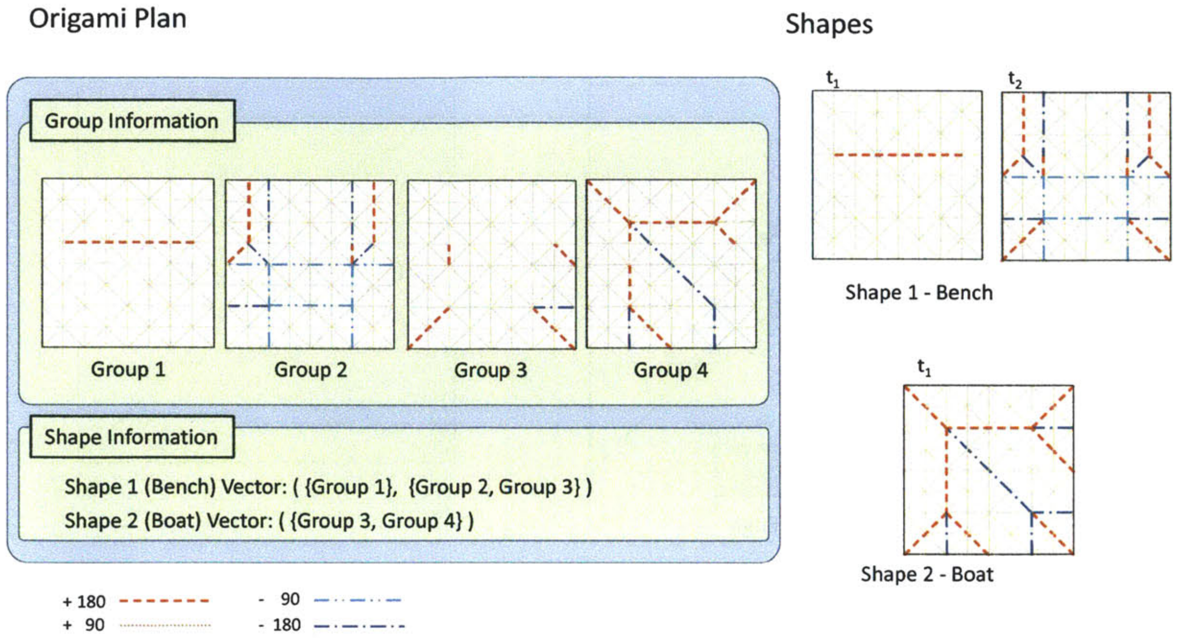


Figure 5-4: Origami plan for bench and boat.

Actuator Model (Sec. 3.1.1)

Definition 5.3.2 *The folding actuator, FA, is expressed with 3-tuple, (Σ, A, δ), where:*

Σ is a finite set of the actuator codes.

A is a finite set of the folding angles.

δ is an angle function, that is, $\delta : \Sigma \rightarrow A$.

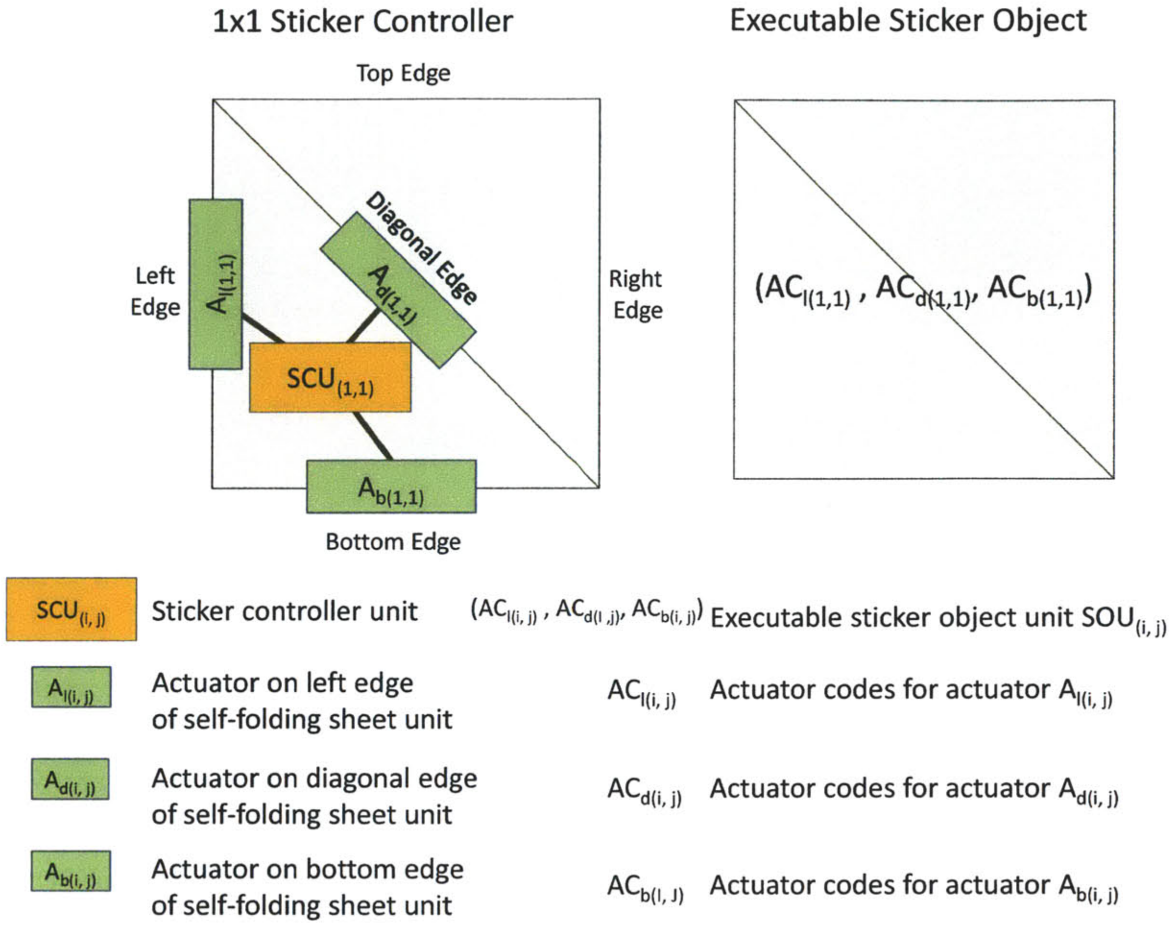

1x1 Sticker Controller
Top Edge
Left Edge
$A_{l(1,1)}$
Diagonal Edge
$A_{d(1,1)}$
Right Edge
$SCU_{(1,1)}$
$A_{b(1,1)}$
Bottom Edge
Executable Sticker Object
$(AC_{l(1,1)}, AC_{d(1,1)}, AC_{b(1,1)})$
$SCU_{(i,j)}$ Sticker controller unit
$A_{l(i,j)}$ Actuator on left edge of self-folding sheet unit
$A_{d(i,j)}$ Actuator on diagonal edge of self-folding sheet unit
$A_{b(i,j)}$ Actuator on bottom edge of self-folding sheet unit
$(AC_{l(i,j)}, AC_{d(I,j)}, AC_{b(i,j)})$ Executable sticker object unit $SOU_{(i,j)}$
$AC_{l(i,j)}$ Actuator codes for actuator $A_{l(i,j)}$
$AC_{d(i,j)}$ Actuator codes for actuator $A_{d(i,j)}$
$AC_{b(I,J)}$ Actuator codes for actuator $A_{b(i,j)}$

Figure 5-5: Executable sticker object for 1×1 sticker controller. (left) Simplified model of 1×1 sticker controller. A sticker controller unit controls three actuators. (right) Corresponding executable sticker object. Each element of the 3-tuple of the object has actuator codes for each actuator.

Executable Sticker Object

An executable sticker object represents an executable sticker with the actuator codes.

An $m \times n$ sticker controller is composed of m columns and n rows of sticker controller units. Each sticker controller unit controls three actuators (Fig. 5-5, 5-6, 5-7).

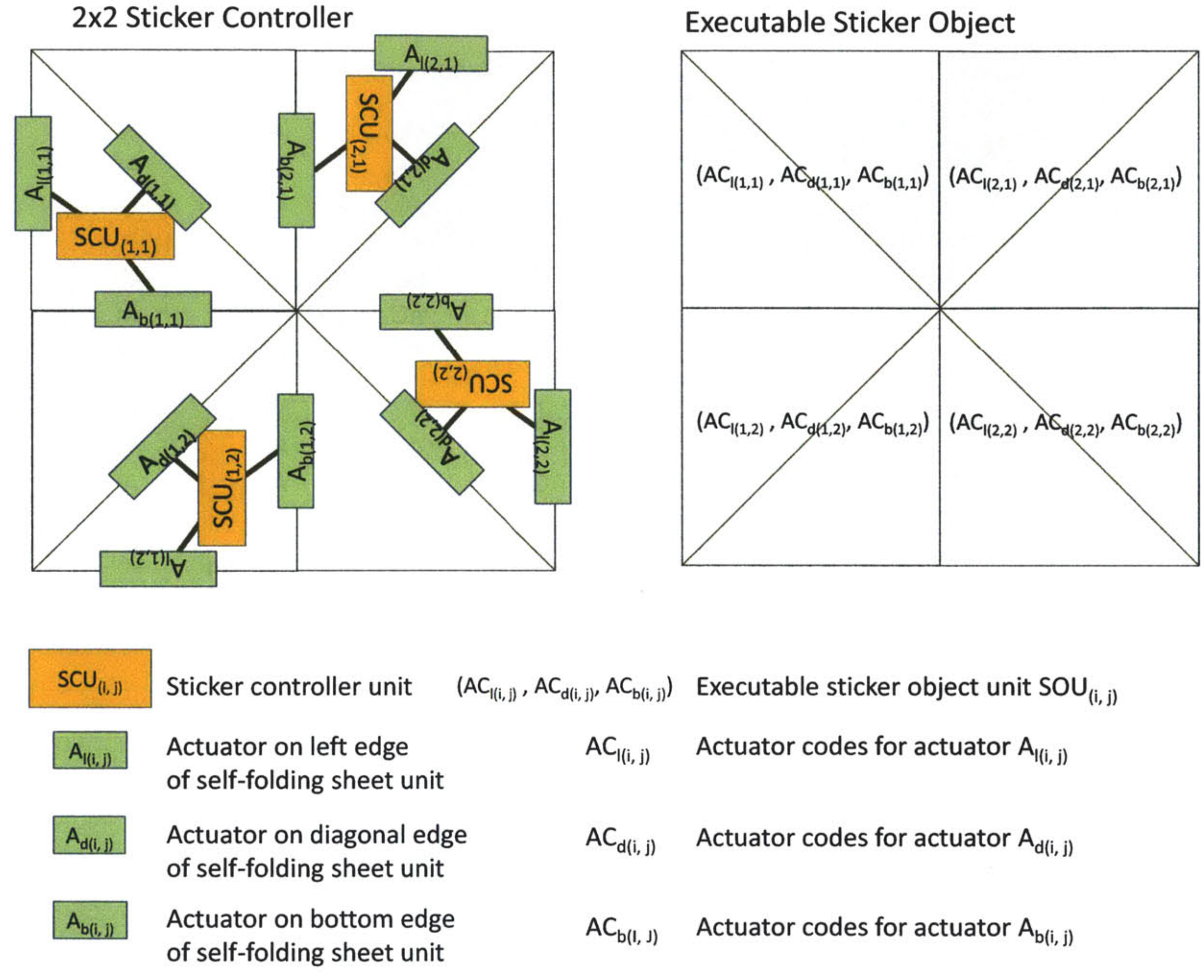

2x2 Sticker Controller
Executable Sticker Object
$A_{l(2,1)}$
$SCU_{(2,1)}$
$A_{d(2,1)}$
$A_{b(2,1)}$
$A_{l(1,1)}$
$A_{d(1,1)}$
$SCU_{(1,1)}$
$A_{b(1,1)}$
$A_{b(2,2)}$
$SCU_{(2,2)}$
$A_{d(2,2)}$
$A_{l(2,2)}$
$A_{d(1,2)}$
$SCU_{(1,2)}$
$A_{b(1,2)}$
$A_{l(1,2)}$
$(AC_{l(1,1)}, AC_{d(1,1)}, AC_{b(1,1)})$
$(AC_{l(2,1)}, AC_{d(2,1)}, AC_{b(2,1)})$
$(AC_{l(1,2)}, AC_{d(1,2)}, AC_{b(1,2)})$
$(AC_{l(2,2)}, AC_{d(2,2)}, AC_{b(2,2)})$
$SCU_{(i,j)}$ Sticker controller unit
$(AC_{l(i,j)}, AC_{d(i,j)}, AC_{b(i,j)})$ Executable sticker object unit $SOU_{(i,j)}$
$A_{l(i,j)}$ Actuator on left edge of self-folding sheet unit
$AC_{l(i,j)}$ Actuator codes for actuator $A_{l(i,j)}$
$A_{d(i,j)}$ Actuator on diagonal edge of self-folding sheet unit
$AC_{d(i,j)}$ Actuator codes for actuator $A_{d(i,j)}$
$A_{b(i,j)}$ Actuator on bottom edge of self-folding sheet unit
$AC_{b(I,J)}$ Actuator codes for actuator $A_{b(i,j)}$

Figure 5-6: Executable sticker object for 2×2 sticker controller. (left) Simplified model of 2×2 sticker controller composed of four 1×1 sticker controllers. Each sticker controller unit on the 1×1 sticker controller controls three actuators. (right) Corresponding executable sticker object. Each element of the 3-tuple of the object has actuator codes for each actuator.

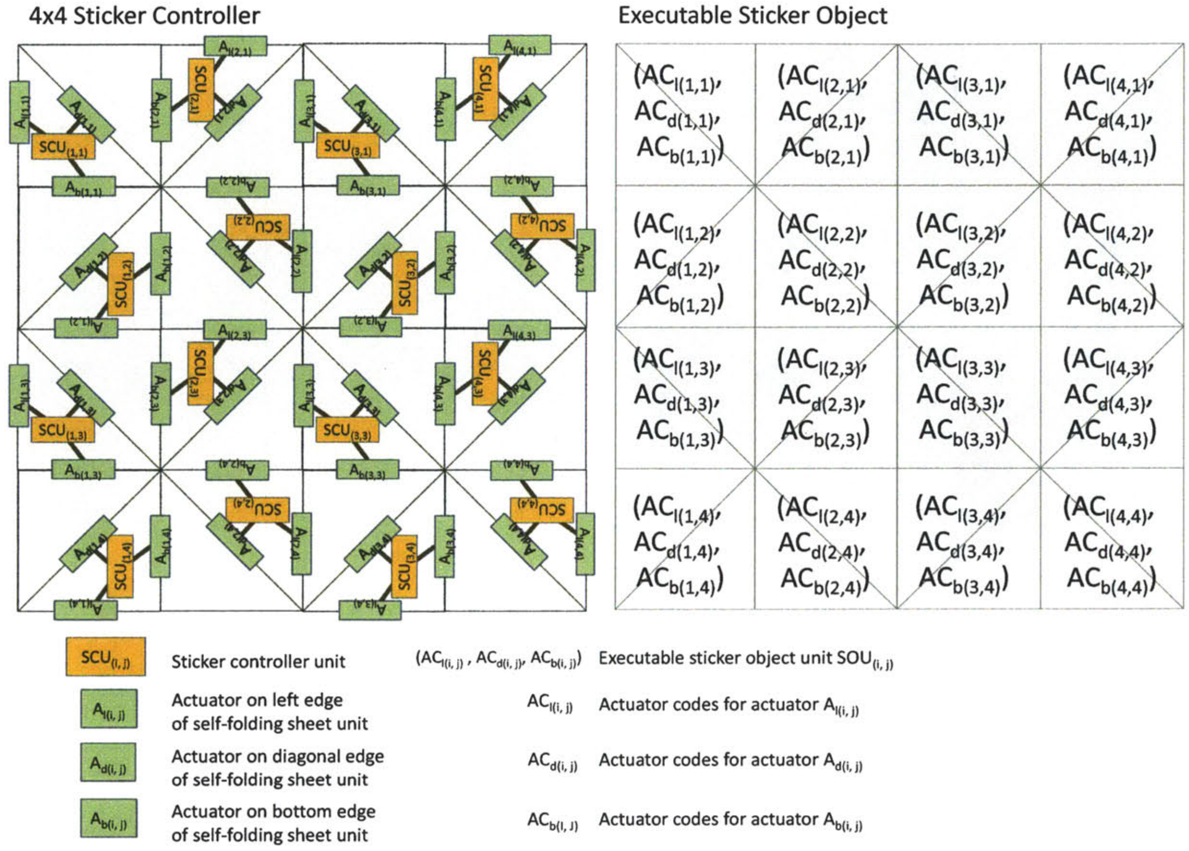

4x4 Sticker Controller
Executable Sticker Object
SCU(1,1) A_l(1,1) A_d(1,1) A_b(1,1)
SCU(2,1) A_l(2,1) A_d(2,1) A_b(2,1)
SCU(3,1) A_l(3,1) A_d(3,1) A_b(3,1)
SCU(4,1) A_l(4,1) A_d(4,1) A_b(4,1)
SCU(1,2) A_l(1,2) A_d(1,2) A_b(1,2)
SCU(2,2) A_l(2,2) A_d(2,2) A_b(2,2)
SCU(3,2) A_l(3,2) A_d(3,2) A_b(3,2)
SCU(4,2) A_l(4,2) A_d(4,2) A_b(4,2)
SCU(1,3) A_l(1,3) A_d(1,3) A_b(1,3)
SCU(2,3) A_l(2,3) A_d(2,3) A_b(2,3)
SCU(3,3) A_l(3,3) A_d(3,3) A_b(3,3)
SCU(4,3) A_l(4,3) A_d(4,3) A_b(4,3)
SCU(1,4) A_l(1,4) A_d(1,4) A_b(1,4)
SCU(2,4) A_l(2,4) A_d(2,4) A_b(2,4)
SCU(3,4) A_l(3,4) A_d(3,4) A_b(3,4)
SCU(4,4) A_l(4,4) A_d(4,4) A_b(4,4)
(AC_l(1,1), AC_d(1,1), AC_b(1,1))
(AC_l(2,1), AC_d(2,1), AC_b(2,1))
(AC_l(3,1), AC_d(3,1), AC_b(3,1))
(AC_l(4,1), AC_d(4,1), AC_b(4,1))
(AC_l(1,2), AC_d(1,2), AC_b(1,2))
(AC_l(2,2), AC_d(2,2), AC_b(2,2))
(AC_l(3,2), AC_d(3,2), AC_b(3,2))
(AC_l(4,2), AC_d(4,2), AC_b(4,2))
(AC_l(1,3), AC_d(1,3), AC_b(1,3))
(AC_l(2,3), AC_d(2,3), AC_b(2,3))
(AC_l(3,3), AC_d(3,3), AC_b(3,3))
(AC_l(4,3), AC_d(4,3), AC_b(4,3))
(AC_l(1,4), AC_d(1,4), AC_b(1,4))
(AC_l(2,4), AC_d(2,4), AC_b(2,4))
(AC_l(3,4), AC_d(3,4), AC_b(3,4))
(AC_l(4,4), AC_d(4,4), AC_b(4,4))
SCU(i, j) Sticker controller unit
A_l(i, j) Actuator on left edge of self-folding sheet unit
A_d(i, j) Actuator on diagonal edge of self-folding sheet unit
A_b(i, j) Actuator on bottom edge of self-folding sheet unit
(AC_l(i, j), AC_d(i, j), AC_b(i, j)) Executable sticker object unit SOU(i, j)
AC_l(i, j) Actuator codes for actuator A_l(i, j)
AC_d(i, j) Actuator codes for actuator A_d(i, j)
AC_b(i, J) Actuator codes for actuator A_b(i, j)

Figure 5-7: Executable sticker object for 4×4 sticker controller. (left) Simplified model of 4×4 sticker controller composed of 16 1×1 sticker controllers. Each sticker controller unit on the 1×1 sticker controller controls three actuators. (right) Corresponding executable sticker object. Each element of the 3-tuple of the object has actuator codes for each actuator.

Definition 5.3.3 *An executable sticker object SO is a set of executable sticker object units $SOU_{i,j}$:*
SO = $\{SOU_{i,j} | SOU_{i,j}$ is an executable sticker object unit, where i is column and j row $\}$ A $SOU_{i,j}$ is 3-tuple (AC_l, AC_d, AC_b), where:
AC_l is actuator codes for a left actuator A_l,
AC_d is actuator codes for a diagonal actuator A_d, and
AC_b is actuator codes for a bottom actuator A_b (Fig. 5-5, 5-6, 5-7, Ch. 4).

Figure 5-5 shows an executable sticker object for 1 × 1 sticker controllers. The sticker controller unit $SCU_{(1,1)}$ controls the left actuator $A_{l(1,1)}$, the diagonal actuator $A_{d(1,1)}$, and the bottom actuator $A_{b(1,1)}$. In Figure 5-5, the diagonal edge is the only foldable edge. it folds according to actuator codes $AC_{d(1,1)}$. The left and bottom edges fold, according to $AC_{l(1,1)}$ and $AC_{d(1,1)}$. Because the left and bottom edges are outline edges, each actuator codes $AC_{l(1,1)}$ and $AC_{d(1,1)}$ are 0s, where 0 is an actuator code for 0 degrees.

Figure 5-6 shows an executable sticker object for a 2 × 2 self-folding sheet, composed of four self-folding sheet units. Each 3-tuple ($AC_{l(i,j)}$, $AC_{d(i,j)}$, $AC_{b(i,j)}$) of the executable sticker object contains information for each sticker controller unit $SCU_{(i,j)}$. Figure 5-7 shows an executable sticker object for a 4×4 self-folding sheet.

Because actuator codes are binary codes, the q^{th} bit of $AC_{x(i,j)}$ is a binary number (0 or 1), where $x \in \{l, d, b\}$. Each q^{th} bit of a $AC_{x(i,j)}$ has an id (sid, q, a), where:
$x \in \{l, d, b\}$,
$sid \in \{C_{l(i,j)}, C_{d(i,j)}, C_{b(i,j)}\}$, and
a is the binary number on q^{th} bit.

Sticker Command Script

Definition 5.3.4 *A sticker command script is a set of pairs (S_i, V_i), where:*
S_i is an origami shape, and
V_i is a command vector (c_1, c_2, ... , c_t), where t is a total folding step. The command vector represents a folding sequence of the origami shape S_i.

A command c_i is a sequence of alphabets g_1 g_2 ... g_n, where the given sticker controller has n input ports of the signal interface. Each input port j connects to all actuators of the actuator group j. The signal is 0 or 1. If a signal g_j of a command c_i is 1, all actuators of an actuator group j is activated in a time step i. If a signal g_j of a command c_i is 0, all actuators of an actuator group j is not activated in a time step i.

Sticker Place Design

A sticker place design of a self-folding sheet contains geometrical and identifying information of each sticker place of the self-folding sheet. A sticker place design is used as a parameter of a self-folding sheet when the sticker linker generates an executable sticker design from given an executable sticker object (Fig. 5-8). Each type of a self-folding sheet has its own sticker place design. In this chapter, we will use sticker place designs for the self-folding sheet model discussed in Chapter 4. Figure 5-9 (a)(b)(c) shows an example model for a 2×2 3-2-1 sticker controller and (d) shows the sticker place design for the model.

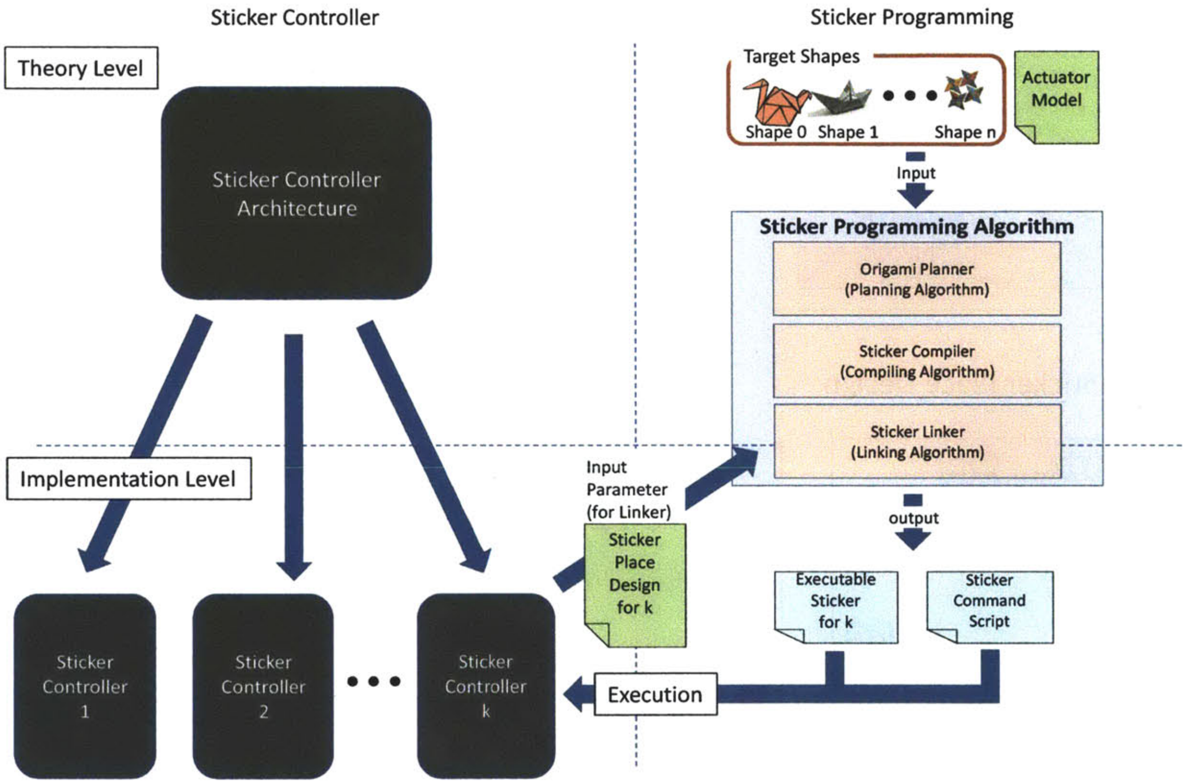


Figure 5-8: Overview of the approach for programming and controlling the self-folding sheet. The sticker place design is used as a parameter of a type of self-folding sheet.

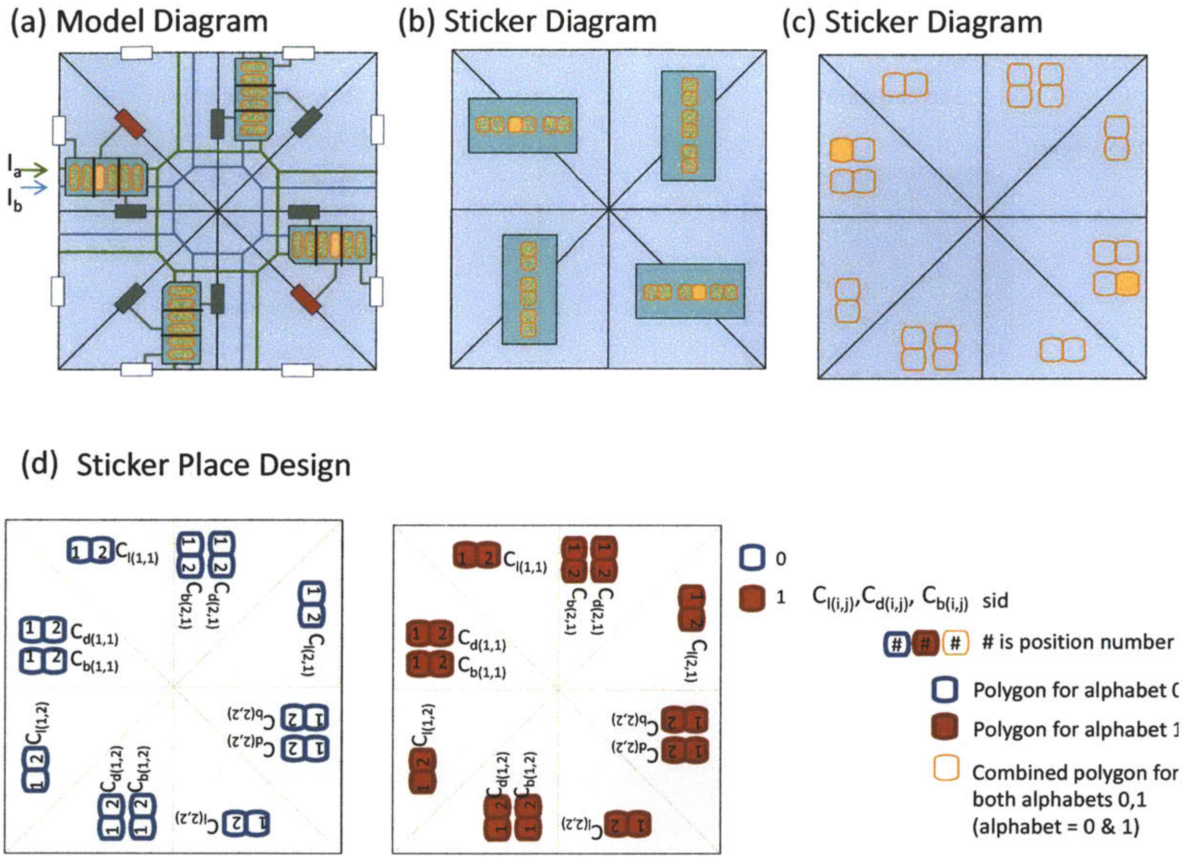


Figure 5-9: Example of a sticker place design for a 2×2 sticker controller. (a)(b)(c) present a same model for the 2×2 sticker controller in three different diagrams. (d) presents the sticker place design for the model. The diagrams (a) and (b) do not contain the locations of the sticker places. The diagram (c) contains the locations of the sticker places.

Definition 5.3.5 *A sticker place design SPD is a set of sticker places SP. SP is a pair (id, P), where:*
id is an identification code and
P is a polygon.
$id = (sid, l, a)$, where:
$sid \in \{C_{l(i,j)}, C_{d(i,j)}, C_{b(i,j)}\}$,
l is a position, and
a alphabet $a \in \{0, 1\}$.
$P = (c, (p_1, p_2, ..., p_n))$, where:
c is a color, and
p_i is a point of the polygon.

The geometrical and identifying information of the sticker place design can be drawn as a diagram. Figure 5-10 shows an example of a sticker place design in two diagrams. (a) and (b) present a same sticker place design composed of 48 sticker places. The 24 sticker places are for an alphabet 0 and the other 24 sticker places are for an alphabet 1. (b) is a simplified diagram of the sticker place design. In Figure 5-10, the id of each sticker place SP is (sid, l, a), where:
$sid \in \{C_{l(i,j)}, C_{d(i,j)}, C_{b(i,j)}\}$,
a position l is the number in the small square, and
an alphabet $a \in \{0, 1\}$.

(c)(d)...(k) are example sticker places with their ids. Let a sticker place be $SP_c = ((C_{l(1,1)}, 1, 0), (blue, (p_1, p_2, p_4, p_3)))$ and let a sticker place be $SP_e = ((C_{l(1,1)}, 1, 1), (red, (p_1, p_2, p_4, p_3)))$. In Figure 5-10 (a), (c) represents SP_c and (e) represents SP_e. The polygons for SP_c and SP_e are on the same location (p_1, p_2, p_4, p_3). However, because of the different colors, they are two different polygons; The polygon of SP_c is the blue while the polygon of SP_e is the red.

When all of the sticker places with alphabet 0 and all of the sticker place with alphabet 1 are at the same locations, we can simplify the sticker place design, as shown in Figure 5-10 (b). In Figure 5-10 (a), all of the sticker places on the left and on the right are on the same location, such as (c) and (e), (d) and (f), or (g) and (h). The only difference between the sticker places on the same location is an alphabet. For instance, the id of (c) is $(C_{l(1,1)}, 1, 0)$ and the id of (e) is $(C_{l(1,1)}, 1, 1)$. Figure 5-10 (b) represents the simplified sticker place design. In the simplified diagram, all polygons have no color but there is the label of the color on the right side of the diagram (b). With the label, we can recognize the alphabet of each sticker place; in (b), the blue is 0 and the red is 1. Each location represents two sticker places with alphabets 0 and 1, such as (i), (j), or (k).

The diagrams contain identifying information (id). By Definition 5.3.5, the id of a sticker place is $(sid,\ l,\ a)$, where $sid \in \{C_{l(i,j)}, C_{d(i,j)}, C_{b(i,j)}\}$, l is a position, and a is an alphabet . In Figure 5-10, (c) and (d) are grouped for a left actuator $A_{l(1,1)}$ and their sid is $C_{l(1,1)}$. (g)'s sid is $C_{l(2,2)}$. The numbers in the polygons are the position. The position of (c) is 1 and the position of (d) is 2. In Figure 5-10(a), the color of each polygon represents alphabets. The label of the color is on the left side of (a). The alphabets of (c) and (d) are 0s, while the alphabets of (e) and (f) are 1s. With the identifying information of the diagram, we can read (c)'s id $(C_{l(1,1)}, 1, 0)$ and (d)'s id $(C_{l(1,1)}, 1, 0)$.

Figure 5-11 shows the same sticker place design with no label. In this chapter, we will use simplified diagrams for the sticker place designs (Fig. 5-11 (b)).

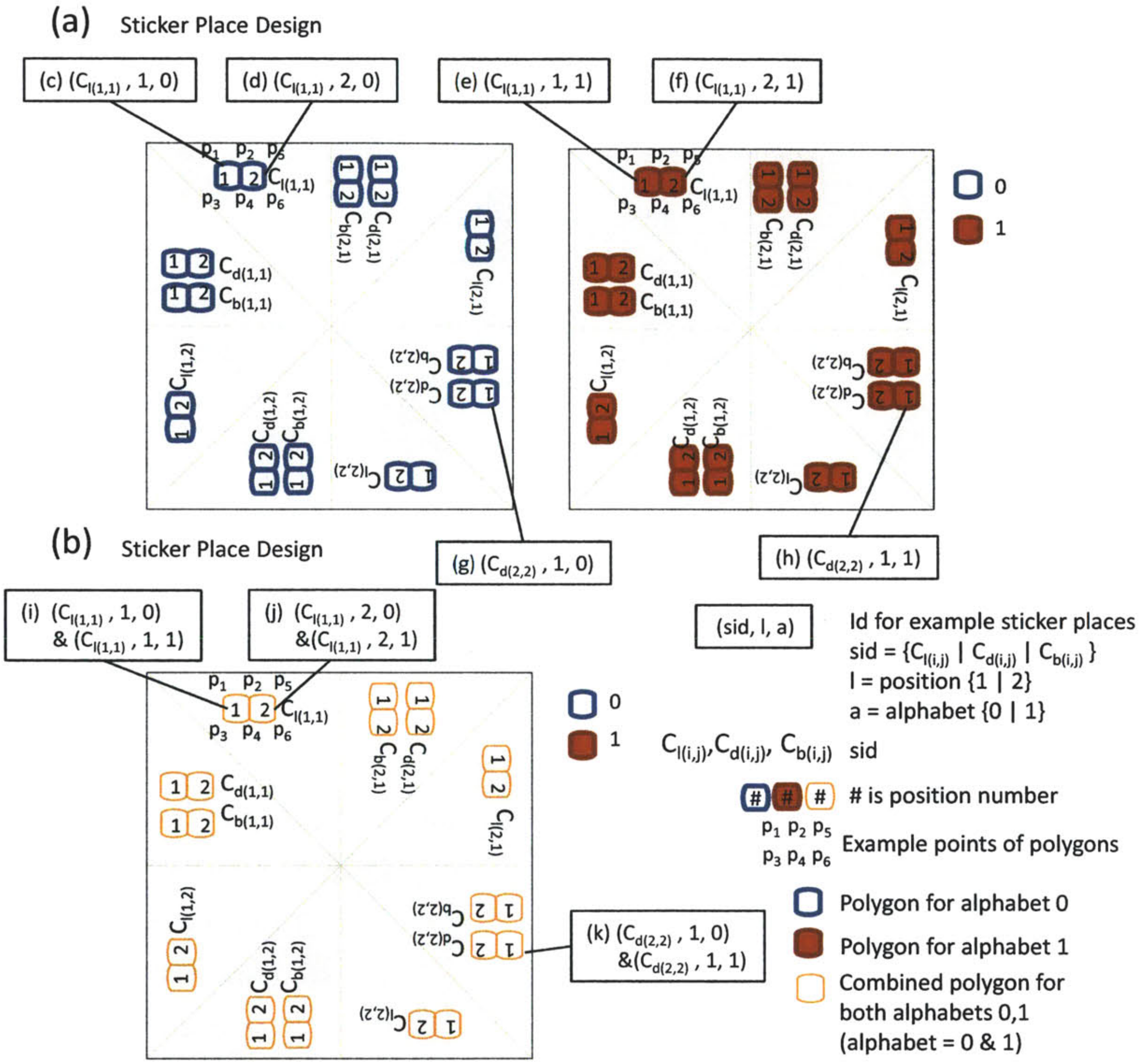


Figure 5-10: Example of sticker place design for 2 × 2 self-folding sheet.
(a) and (b) present a same sticker place design. (a) is a detailed diagram for the sticker place design. (b) is a simplified diagram for the sticker place design. The boxes of (c)(d)...(k) have the ids of the sticker places.

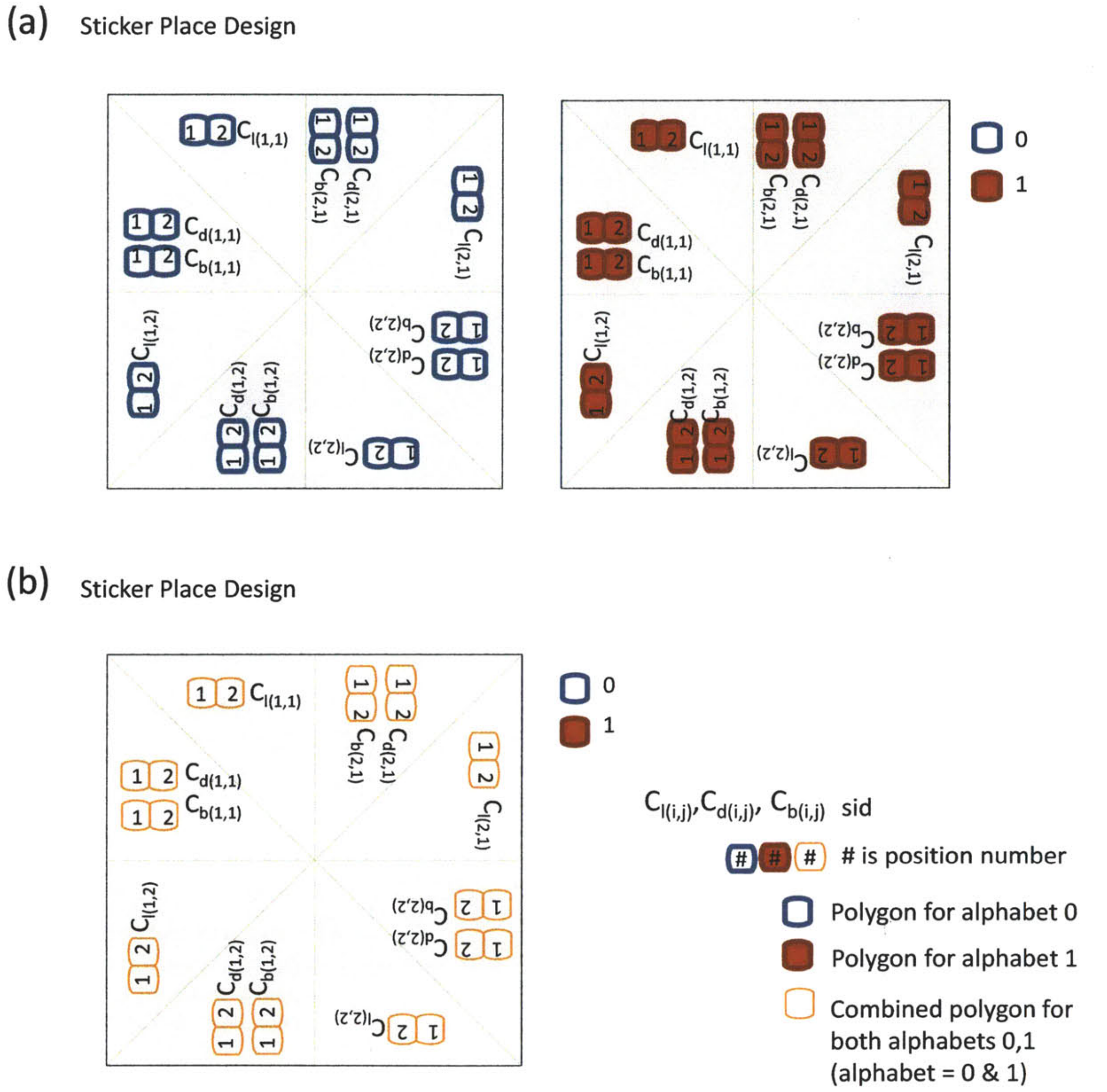


Figure 5-11: Example of sticker place design with no label for 2 × 2 self-folding sheet. (a) and (b) present a same sticker place design. (a) is a detailed diagram for the sticker place design. (b) is a simplified diagram for the sticker place design.

Executable Sticker Design

An executable sticker design contains graphical information about an executable sticker.

Definition 5.3.6 *An executable sticker design ESD is a set of polygon graphs P. $P = (c, (p_1, p_2, ..., p_n))$, where c is a color and p_i is a point of the polygon.*

Figure 5-12 shows an example of an executable sticker design where as Figure 5-13 shows snapshots of an example executable sticker design in .dxf file (CAD diagram).

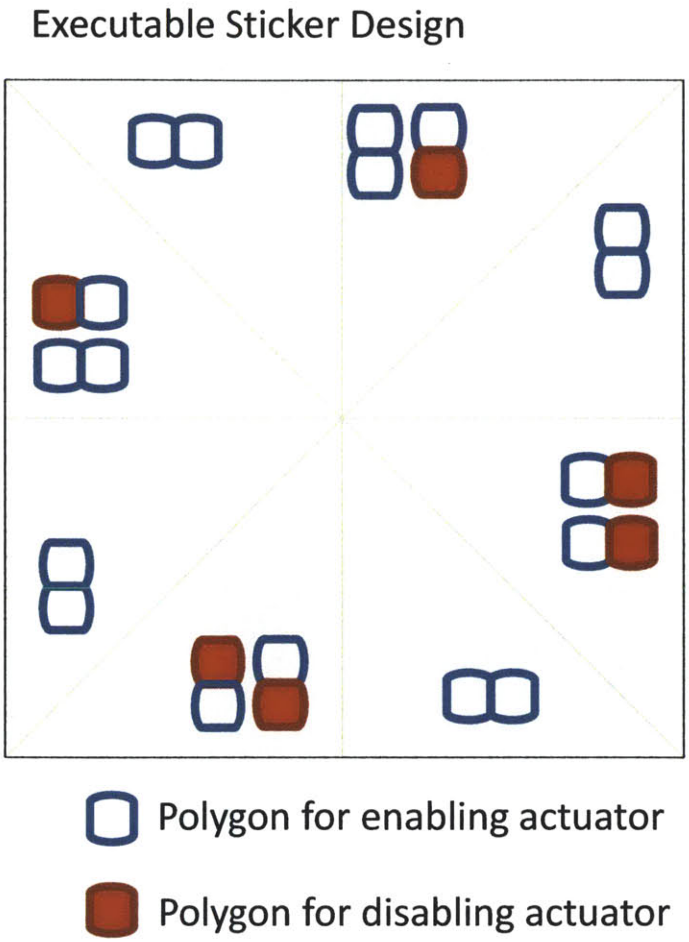

Executable Sticker Design
Polygon for enabling actuator
Polygon for disabling actuator

Figure 5-12: Example of executable sticker design. The executable sticker design is the result of a sticker linking example discussed in Section 5.3.3. The red square is a sticker place enabling its actuator. The blue square is a sticker place disabling its actuator.

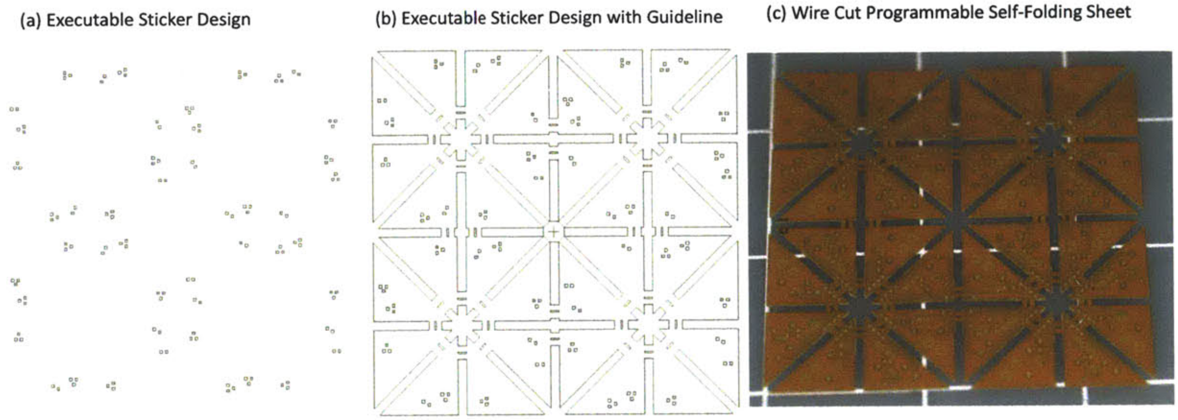

(a) Executable Sticker Design
(b) Executable Sticker Design with Guideline
(c) Wire Cut Programmable Self-Folding Sheet

Figure 5-13: Example of an executable sticker design for an implemented self-folding sheet. (a) Executable sticker design for two shapes (a space-shuttle and a pyramid). By manually computing the sticker linker, we generate the executable sticker design. (b) Executable sticker design with guide line. The guideline is outline of the W-sheet(c). (a) and (b) are drawn as .dxf files (CAD diagrams) for Diode-Pumped Sold State (DPSS) Laser Micromachining System (Custom Build, at the Micro Robotics Lab, Harvard University). (c) 4 × 4 wire cut programmable self-folding sheet (W-sheet) with no program. The W-sheet embeds a specially designed circuit. We program the W-sheet by making small square-shape holes according to (a).

Sticker Compiler (Fig. 5-15)

1. Given the group information of an origami plan and an actuator model of a fixed self-folding sheet, convert each angle of the edges to its corresponding actuator codes.
2. Combine the actuator codes of all groups of each edge.
3. Construct an executable sticker object by collecting the combined actuator codes on the edges (Fig. 5-16).
4. Construct a sticker command script by converting the folding sequence of each shape (the vectors of the shape information of the origami plan) into a sequence of commends.
5. Output the executable sticker object and the sticker command script.

Figure 5-14: Algorithmic overview of sticker compiler.

5.3.2 Sticker Compiler

Given the group information of an origami plan and an actuator model of a self-folding sheet, the sticker compiler generates an executable sticker object and a sticker command script. Figure 5-14 shows the five step process overview. Figure 5-15 shows an example of the algorithm generating an executable sticker object for a 2×2 sticker controller. In this example, the origami plan contains Shape 1 (the nose of a boat) and Shape 2 (a triangle). The origami plan in Figure 5-15 contains two groups in the group information and two folding sequences in the shape information. To achieve Shape 1 according to the plan the self-folding sheet must fold the edges of Group 1. To achieve Shape 2, the self-folding sheet must fold the edges of Group 2.

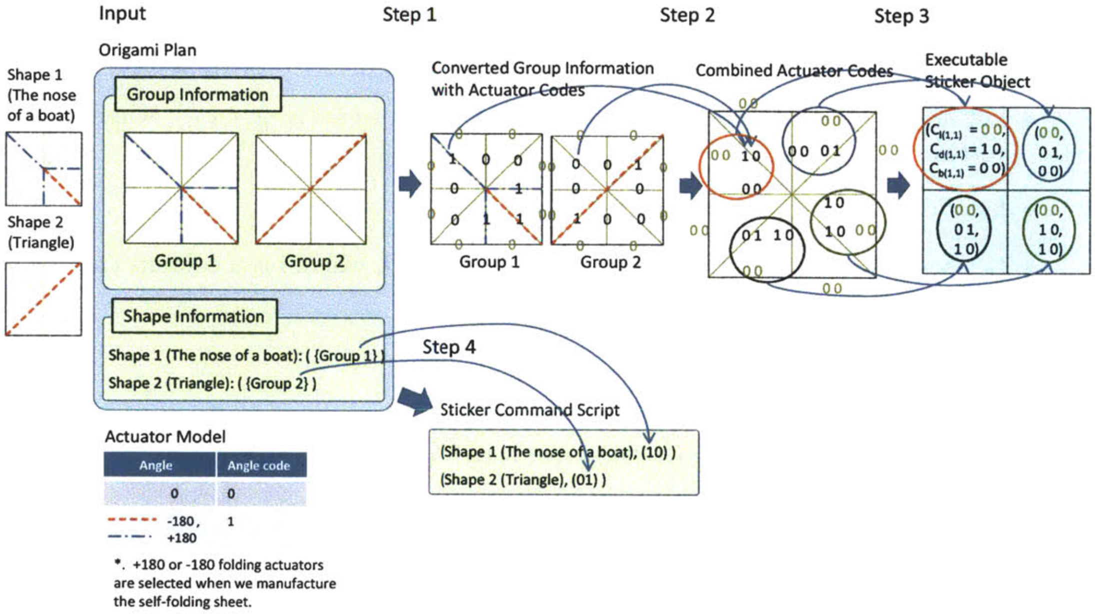


Figure 5-15: Compiling a sticker for 2 × 2 sticker controller for the nose of a boat and a triangle (Overview of Fig. 5-14). Given origami plan (input), the sticker compiler constructs an executable sticker object and a sticker command script. Each step of the example is shown in much detail in Figures 5-17, 5-18, 5-19, 5-20.

Constructing Executable Sticker Object (Step 3 of Sticker Compiler)

- Given the converted actuator codes of each edge (From step 2 in Fig. 5-14).
- Let $ac_{(i,j)-(k,l)}$ be a converted actuator code on an edge $(v_{(i,j)}, v_{(k,l)})$, where v is a vertex.
- Let SO be an executable sticker object.
- Let $sou_{i,j}$ be a executable sticker object unit of SO, where i is a column and j is a row.
- For each column i and row j:
  1. If i = odd and j = odd,
     $sou_{i,j} \leftarrow (ac_{(i,j)-(i,j+1)}, ac_{(i,j)-(i+1,j+1)}, ac_{(i,j+1)-(i+1,j+1)})$.
  2. If i = even and j = odd,
     $sou_{i,j} \leftarrow (ac_{(i,j)-(i+1,j)}, ac_{(i+1,j)-(i,j+1)}, ac_{(i,j)-(i,j+1)})$.
  3. If i = odd and j = even,
     $sou_{i,j} \leftarrow (ac_{(i,j+1)-(i+1,j+1)}, ac_{(i+1,j)-(i,j+1)}, ac_{(i+1,j)-(i+1,j+1)})$.
  4. If i = even and j = even,
     $sou_{i,j} \leftarrow (ac_{(i+1,j)-(i+,j+1)}, ac_{(i,j)-(i+1,j+1)}, ac_{(i,j+1)-(i+1,j)})$.

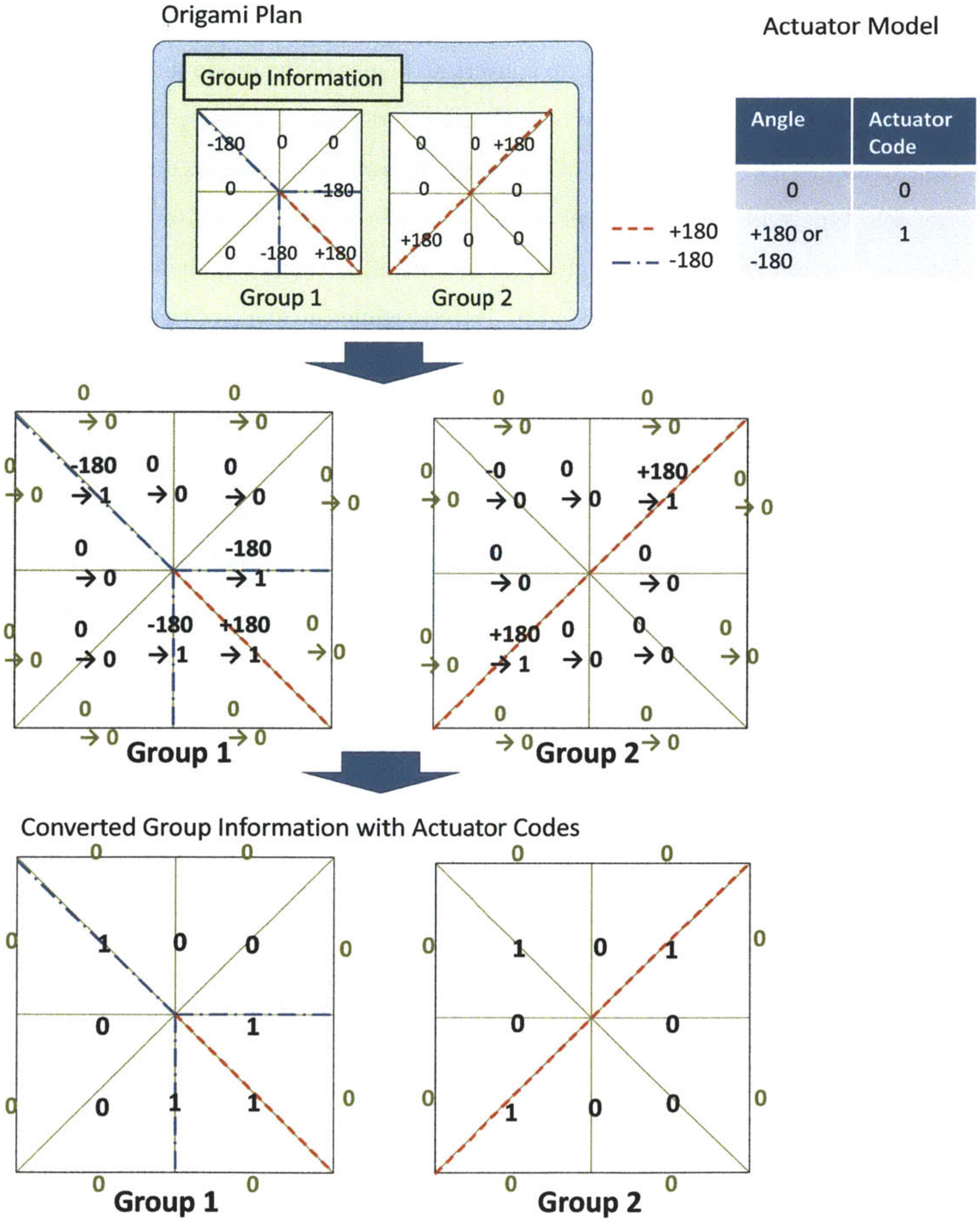
Figure 5-16: Constructing executable sticker object. Details of sticker compiler step 3.

Generating the Executable Sticker Object

The first step (Step 1 in Figure 5-14) is to convert all angles of the group information to their corresponding actuator codes. Figure 5-17 shows the details of the step 1 of the example. Each edge of all groups (given as part of the group information) contains an angle (in Figure 5-17, the line type and color represent an angle of the edge). The given actuator model contains the function table of the angles and the actuator codes. In the origami plan, the angle of the top-left edge of Group 1 is +180 degrees, while the angle of the top-left edge the Group 2 is 0 degrees. The algorithm converts the angle +180 degrees to the actuator code 1 and the angle 0 degrees to the actuator code 0, following the actuator model.

The second step is to combine all actuator codes of each edge. The algorithm combines the actuator codes of each edge into the combined actuator codes (Fig. 5-18). An actuator code of the top-left edge of Group 1 is 1 while an actuator code of the top-left edge of Group 2 is 0. The algorithm combines these two actuator codes into the combined actuator codes 10 for the top-left edge

The third step is to construct an executable sticker object by collecting the combined actuator codes of the edges. Figure 5-16 shows the detailed process of the step 3. Figure 5-19 (a) shows detailed process of the step 3. Each 3-tuple of the executable sticker object contains information for a sticker controller unit ((b), (c)).



Angle	Actuator Code
0	0
+180 or -180	1

Figure 5-17: Compiling a sticker object for 2 × 2 sticker controller the nose of a boat and a triangle (Step 1 on Fig. 5-14). Given the group information (Group1 and Group2) of the origami plan and the actuator model (top), the algorithm converts the angle of each edge to the actuator code (middle). The result of step 1 is the converted group information with actuator codes (bottom).

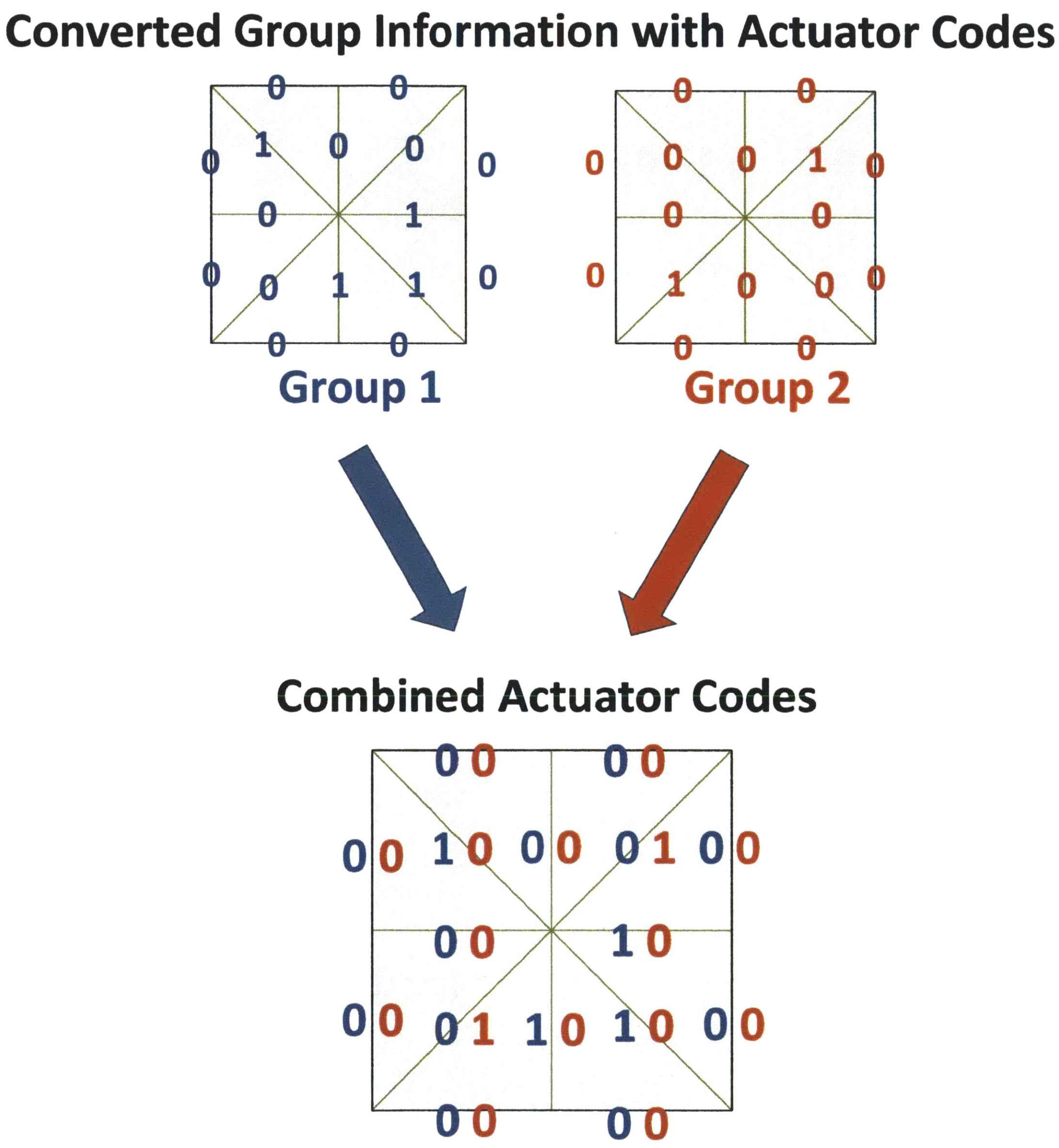
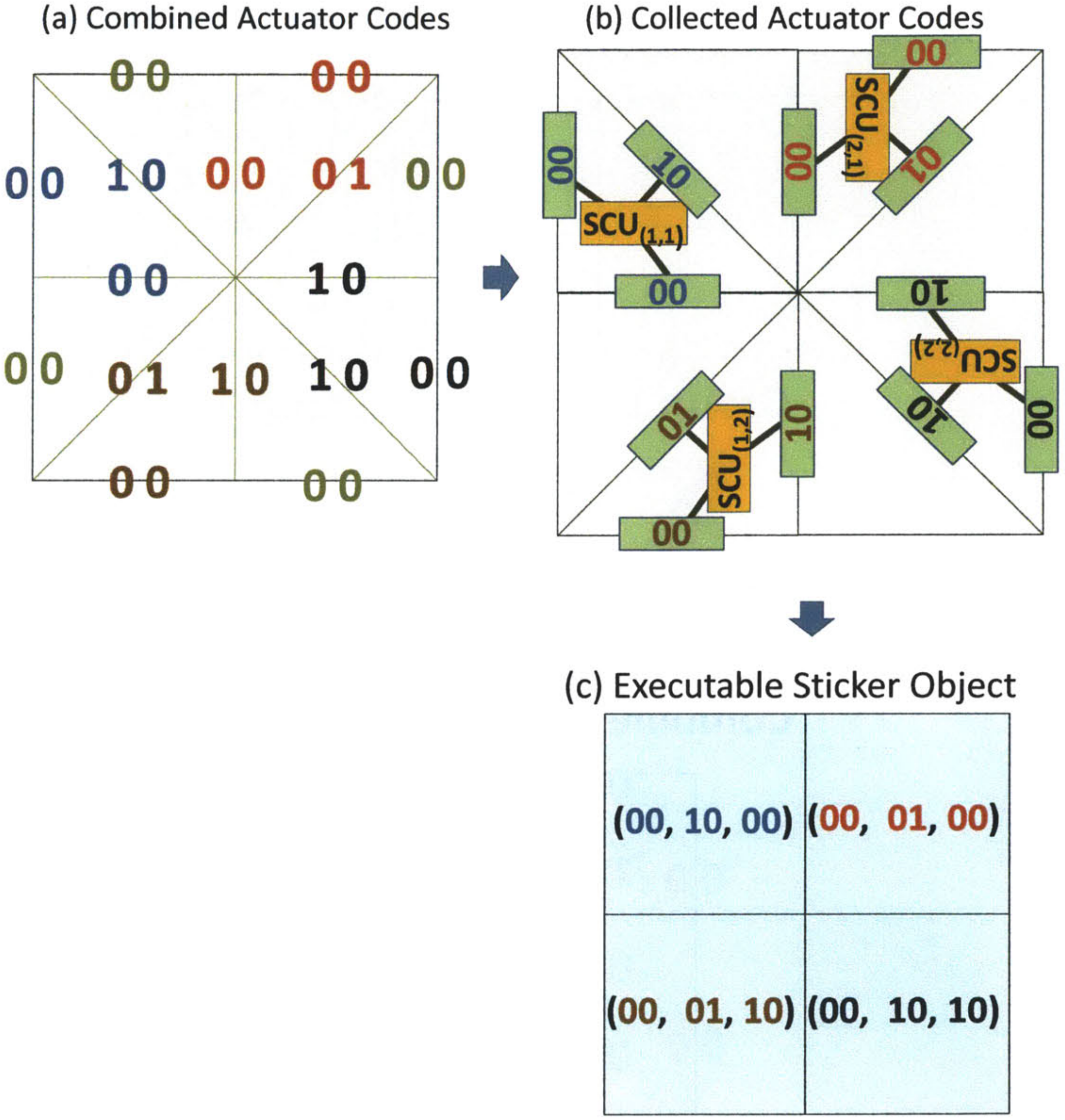


Figure 5-18: Compiling a sticker object for 2 × 2 sticker controller for the nose of a boat and a triangle (Step 2 on Fig. 5-14). Given the converted group information with actuator codes (top) from step 1 (Fig. 5-17), the algorithm combines the actuator codes (bottom). The result of step 2 is the combined actuator codes (bottom). The blue actuator codes are from Group 1. The red actuator codes are from Group 2.


(a) Combined Actuator Codes

(b) Collected Actuator Codes

(c) Executable Sticker Object

(00, 10, 00)	(00, 01, 00)
(00, 01, 10)	(00, 10, 10)

Figure 5-19: Compiling a sticker object for 2 × 2 sticker controller for the nose of a boat and a triangle (Step 3 on Fig. 5-14, Fig. 5-16). Given the combined actuator codes (a) from step 2 (Fig. 5-18), the algorithm builds the executable sticker object (c). (b) shows corresponding actuator codes for each sticker controller unit $SCU_{(i,j)}$. The blue actuator codes are for the $SCU_{(1,1)}$. The red actuator codes are for the $SCU_{(2,1)}$. The brown actuator codes are for the $SCU_{(1,2)}$. The black actuator codes are for the $SCU_{(2,2)}$.

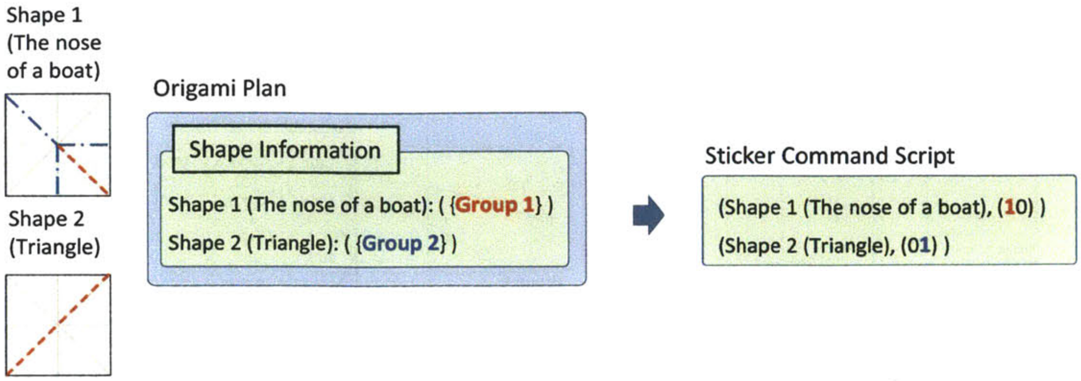
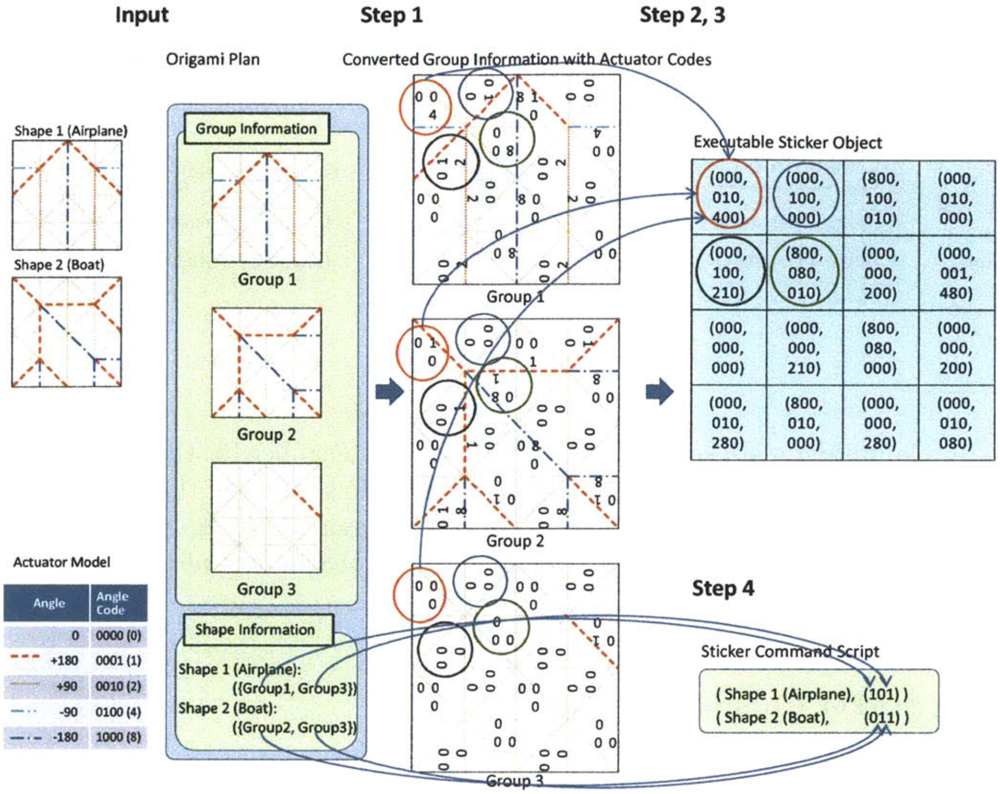

Shape 1
(The nose
of a boat)
Shape 2
(Triangle)
Origami Plan
Shape Information
Shape 1 (The nose of a boat): ({Group 1})
Shape 2 (Triangle): ({Group 2})
Sticker Command Script
(Shape 1 (The nose of a boat), (10))
(Shape 2 (Triangle), (01))

Figure 5-20: Compiling a sticker object for 2 × 2 sticker controller for the nose of a boat and a triangle (Step 4 on Fig. 5-14). Shape 1 of the shape information of the given origami plan (Fig. 5-15) is converted to the first line of the sticker command script. Shape 2 of the shape information of the given origami plan is converted to the second line of the sticker command script.

Generating Sticker Command Script

In the forth step, the sticker compiler converts shape information of an origami plan into a sticker command script by replacing the group names to binary codes (or signals for the signal interface). In Figure 5-20, because each Shape 1 (The nose of a boat) and Shape 2 (a triangle) is folded in one time step, each vector contains one command. Each alphabet of the command represents groups folding simultaneously during a particular time step. The command for Shape 1 is 10 while the command for Shape 2 is 01 (by folding only Group 1, the self-folding sheet transforms itself into Shape 1 and similarly, Group 2 transforms into Shape 2).

A sticker command script in Figure 5-20 contains the pair (*Shape* 1, (10)). The top-left 3-tuple of an executable sticker object is (00, 10, 00) (Fig. 5-19). When the sticker controller receives the command 10, the top-left sticker controller unit sends actuator codes 0, 1, and 0 to their corresponding – left, diagonal, and bottom – actuators, respectively. In the meantime, the top-right sticker controller unit sends actuator codes 0, 0, and 0 to their corresponding actuators respectively. The bottom-left sticker controller unit sends actuator codes 0, 0, and 1 (and the bottom-right sends 0, 1, and 1) to their corresponding actuators,



Actuator Model

Angle	Angle Code
0	0000 (0)
+180	0001 (1)
+90	0010 (2)
-90	0100 (4)
-180	1000 (8)

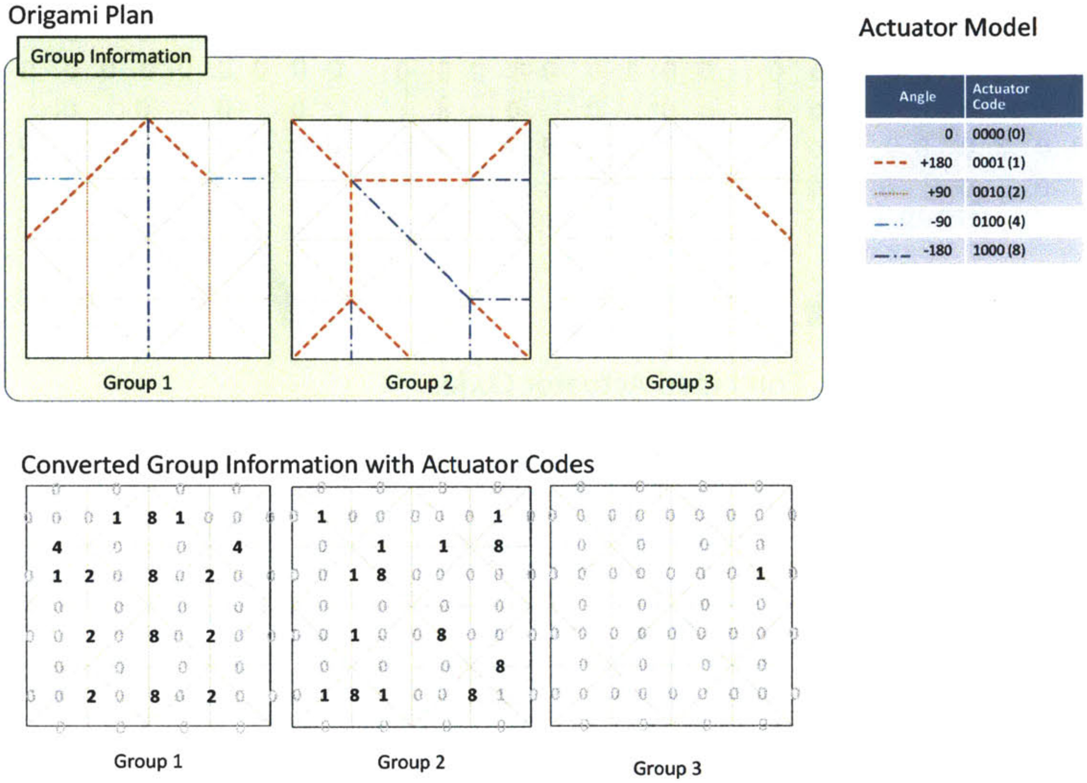
Executable Sticker Object

(000, 010, 400)	(000, 100, 000)	(800, 100, 010)	(000, 010, 000)
(000, 100, 210)	(800, 080, 010)	(000, 000, 200)	(000, 001, 480)
(000, 000, 000)	(000, 000, 210)	(800, 080, 000)	(000, 000, 200)
(000, 010, 280)	(800, 010, 000)	(000, 000, 280)	(000, 010, 080)

Figure 5-21: Compiling a sticker object for 4 × 4 sticker controller for an airplane and a boat (Overview of Fig. 5-14).

respectively. Then, the self-folding sheet transforms itself into Shape 1 – the nose of a boat. (All sticker controller units simultaneously send these actuator codes.)

Figure 5-21 shows an overview of another sticker compiling example. The goal shapes of the example are an airplane and a boat. Figures 5-22, 5-23, 5-24, and 5-25 show details of each step.



	Angle	Actuator Code
	0	0000 (0)
- - -	+180	0001 (1)
········	+90	0010 (2)
— · ·	-90	0100 (4)
— · —	-180	1000 (8)

Figure 5-22: Compiling a sticker object for 4 × 4 sticker controller for an airplane and a boat (Step 1 on Fig. 5-14). Given the origami plan and the actuator model, the algorithm generates a converted group information with actuator codes. The bold numbers are converted numbers from the angles that is not 0 degree. The group information of the origami plan has information about the angle of each edge; each type of line represents an angle (see the actuator model). The actuator model defines the angles and their corresponding actuator codes; hex codes are in ( ).

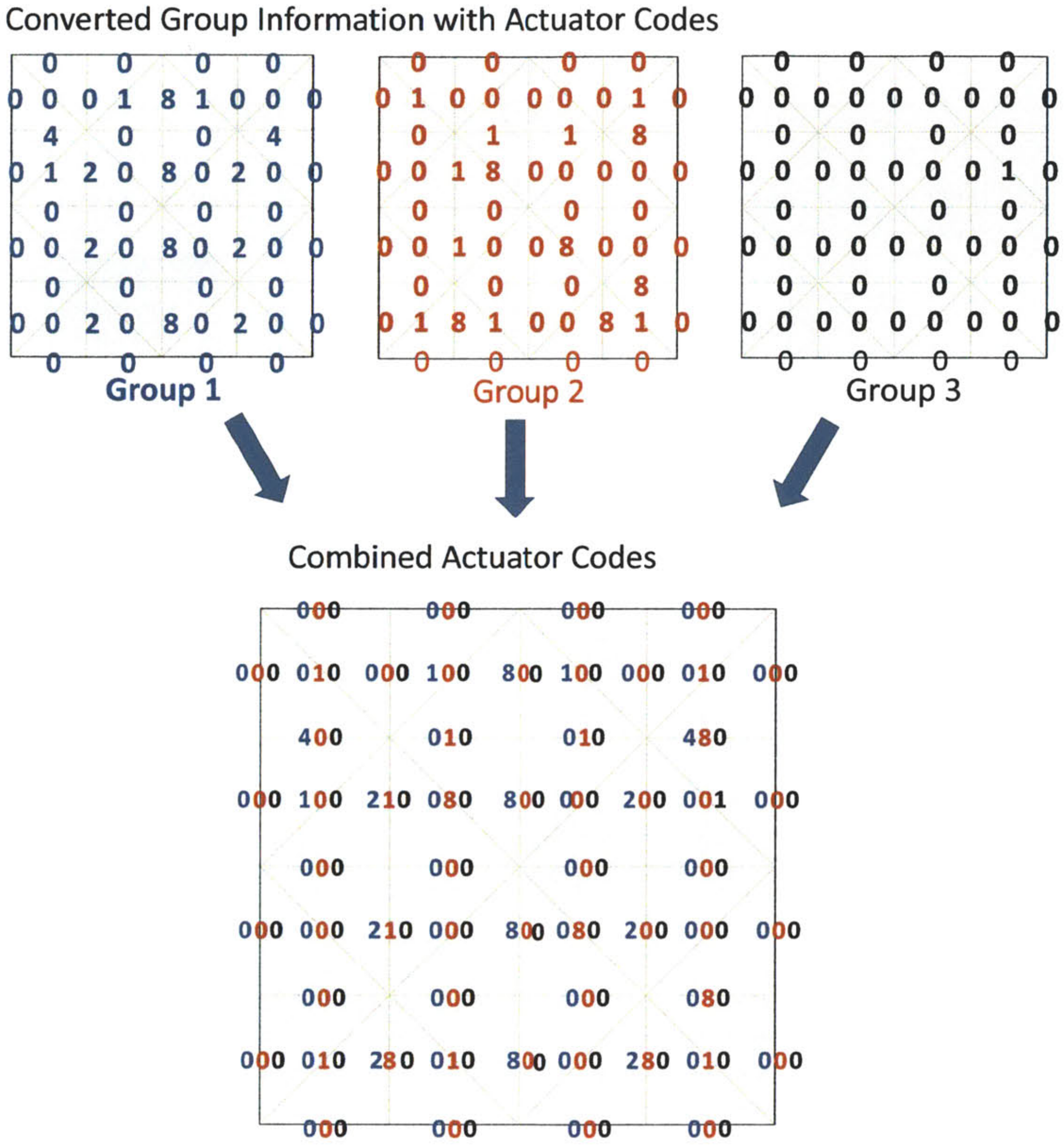
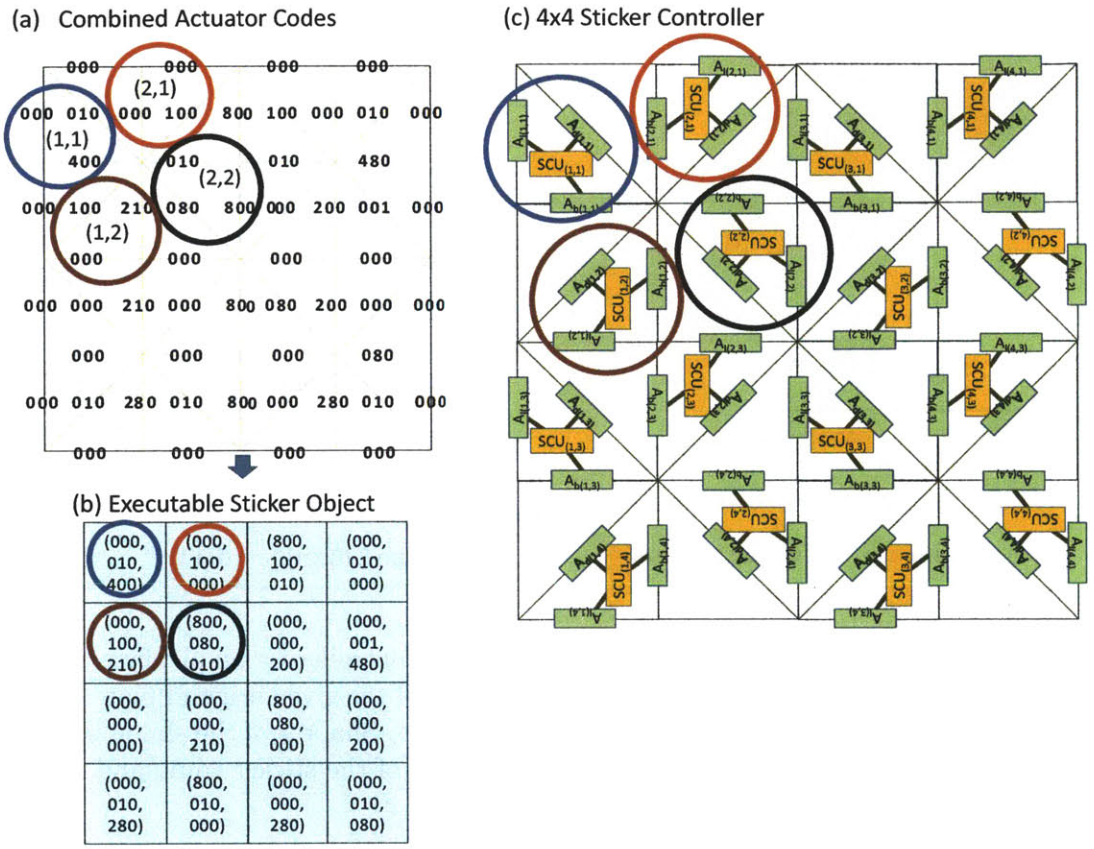
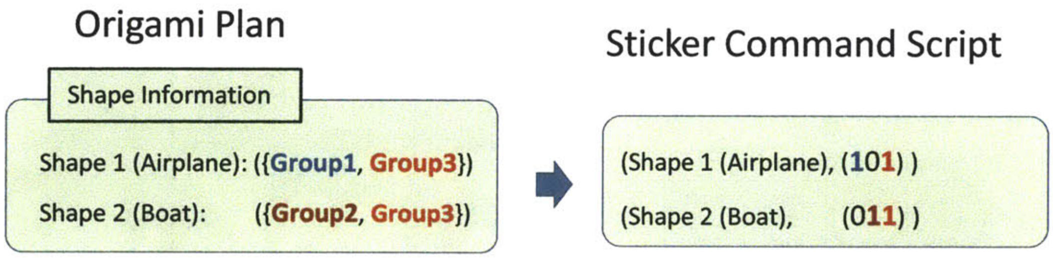


Figure 5-23: Compiling a sticker object for 4 × 4 sticker controller for an airplane and a boat (Step 2 on Fig. 5-14). Given the converted group information with actuator codes (Fig. 5-22), the algorithm combines actuator codes of each group. The red numbers of combined actuator codes are from Group 1. The blue numbers of combined actuator codes are from Group 2. The black numbers of combined actuator codes are from Group 3. The numbers are the hex codes of the actuator codes (Fig. 5-22).



(000, 010, 400)	(000, 100, 000)	(800, 100, 010)	(000, 010, 000)
(000, 100, 210)	(800, 080, 010)	(000, 000, 200)	(000, 001, 480)
(000, 000, 000)	(000, 000, 210)	(800, 080, 000)	(000, 000, 200)
(000, 010, 280)	(800, 010, 000)	(000, 000, 280)	(000, 010, 080)

Figure 5-24: Compiling a sticker object for 4×4 sticker controller for an airplane and a boat (Step 3 on Fig. 5-14, Fig. 5-16). Given the combined actuator codes (a), the algorithm generates the executable sticker object (b). (c) is a simplified model of a 4×4 sticker controller. (c) shows the three corresponding actuators (or actuator codes) for each sticker controller unit $SCU_{(i,j)}$. The blue circles in (a)(b)(c) are for $SCU_{(1,1)}$. The red circles in (a)(b)(c) are for $SCU_{(2,1)}$. The brown circles in (a)(b)(c) are for $SCU_{(1,2)}$. The black circles in (a)(b)(c) are for $SCU_{(2,2)}$.



Origami Plan

Shape Information

Shape 1 (Airplane): ({Group1, Group3})

Shape 2 (Boat): ({Group2, Group3})

Sticker Command Script

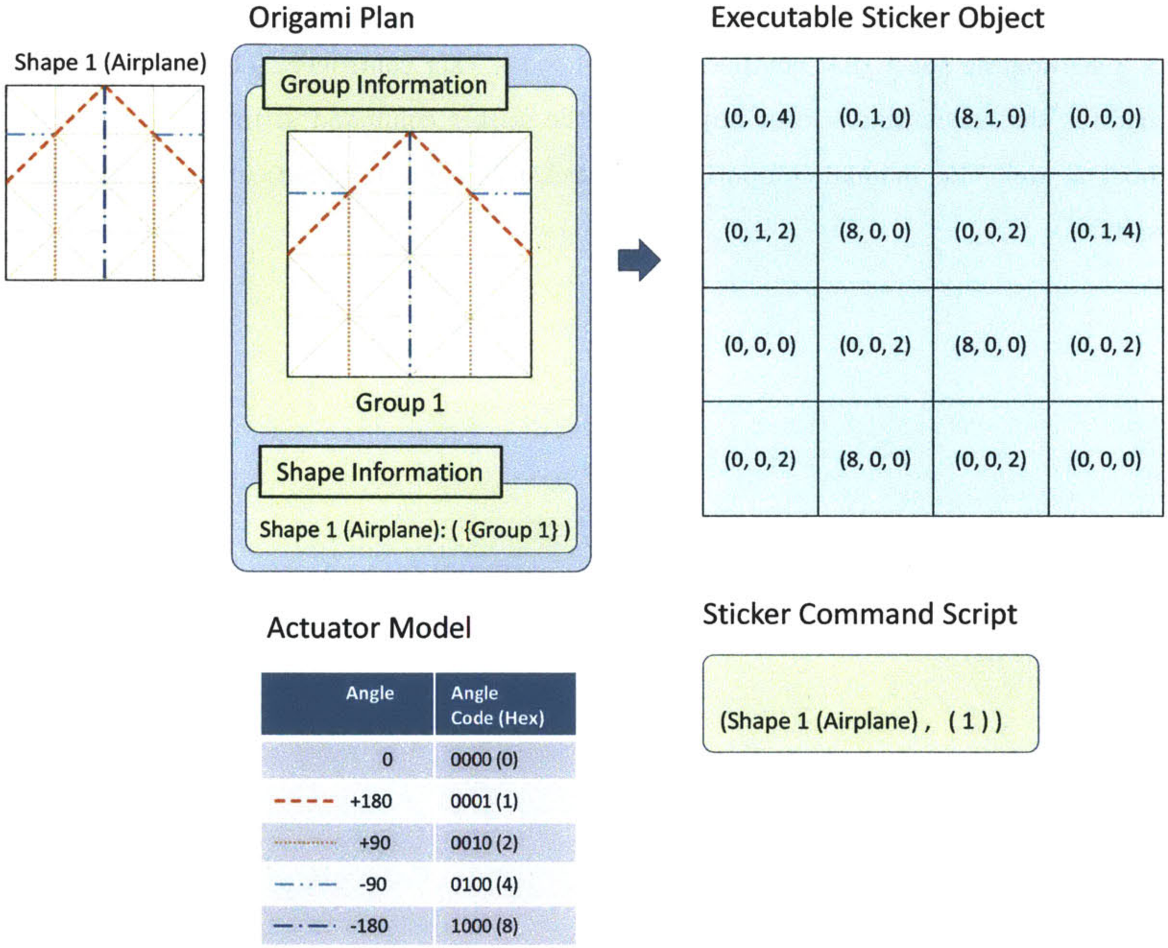
(Shape 1 (Airplane), (101))

(Shape 2 (Boat), (011))

Figure 5-25: Compiling a sticker object for 4 × 4 sticker controller for an airplane and a boat (Step 4 on Fig. 5-14). Shape 1 of the shape information of the given origami plan (Fig. 5-21) is converted to the first line of the sticker command script. Shape 2 of the shape information of the given origami plan is converted to the second line of the sticker command script.

The sticker compiler works for simpler or more complex origamis. Figure 5-26 shows a result of the sticker compiler for a single shape, an airplane. The sticker compiler achieves to generate the executable sticker object and the sticker command script. Figure 5-27 shows a result of the sticker compiler for three shapes, a boat, a tray, and a table.

The sticker compiler is scalable. Figure 5-28 shows a result of the sticker compiler for a 8 × 8 self-folding sheet that composed of 64 1 × 1 sticker controller. The sticker compiler generates the executable sticker object and the sticker command script for a bench and a boat as well. By manual computing, the sticker compiler automatically generates these results.



Executable Sticker Object

(0, 0, 4)	(0, 1, 0)	(8, 1, 0)	(0, 0, 0)
(0, 1, 2)	(8, 0, 0)	(0, 0, 2)	(0, 1, 4)
(0, 0, 0)	(0, 0, 2)	(8, 0, 0)	(0, 0, 2)
(0, 0, 2)	(8, 0, 0)	(0, 0, 2)	(0, 0, 0)

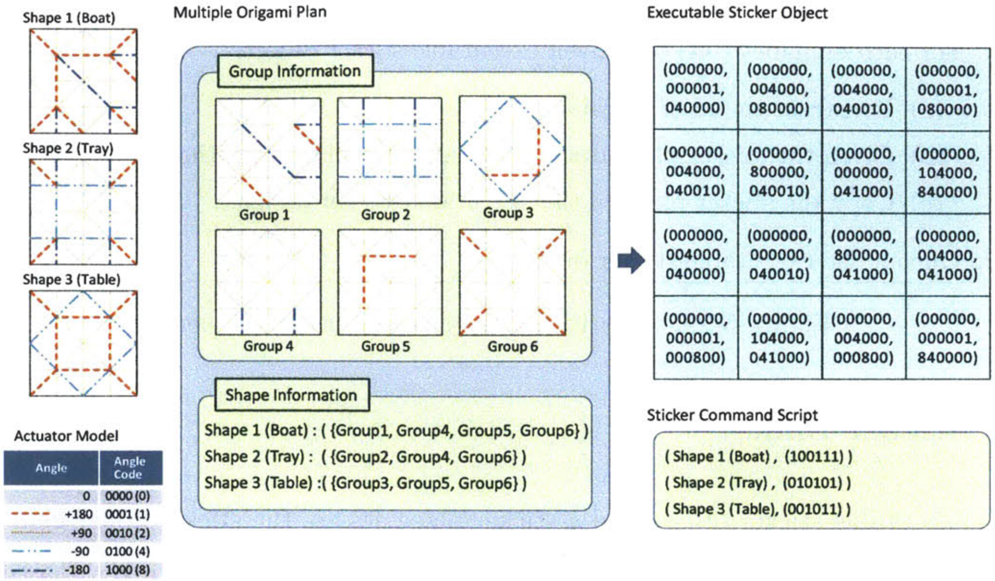
Actuator Model

Angle	Angle Code (Hex)
0	0000 (0)
- - - - +180	0001 (1)
·········· +90	0010 (2)
—··— -90	0100 (4)
—·—· -180	1000 (8)

Sticker Command Script

(Shape 1 (Airplane) , (1))

Figure 5-26: Result of the sticker compiler for a single shape, an airplane. The actuator codes of the executable sticker object is hex. The hex actuator codes of the actuator model are in ( ).



Shape Information

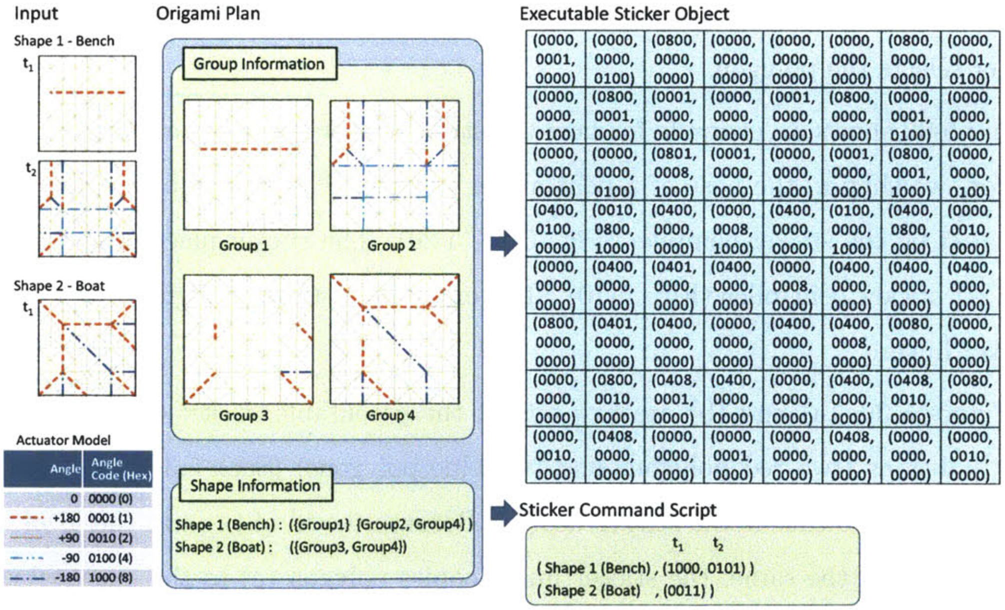
Shape 1 (Boat) : ({Group1, Group4, Group5, Group6})
Shape 2 (Tray) : ({Group2, Group4, Group6})
Shape 3 (Table) :({Group3, Group5, Group6})

(000000, 000001, 040000)	(000000, 004000, 080000)	(000000, 004000, 040010)	(000000, 000001, 080000)
(000000, 004000, 040010)	(000000, 800000, 040010)	(000000, 000000, 041000)	(000000, 104000, 840000)
(000000, 004000, 040000)	(000000, 000000, 040010)	(000000, 800000, 041000)	(000000, 004000, 041000)
(000000, 000001, 000800)	(000000, 104000, 041000)	(000000, 004000, 000800)	(000000, 000001, 840000)

(Shape 1 (Boat) , (100111))
(Shape 2 (Tray) , (010101))
(Shape 3 (Table), (001011))

Angle	Angle Code
0	0000 (0)
+180	0001 (1)
+90	0010 (2)
-90	0100 (4)
-180	1000 (8)

Figure 5-27: Result of the sticker compiler for three shapes, a boat, a tray and a table. The actuator codes of the executable sticker object is hex. The hex actuator codes of the actuator model are in ( ).



Shape Information

Shape 1 (Bench) : ({Group1} {Group2, Group4})
Shape 2 (Boat) : ({Group3, Group4})

(0000, 0001, 0000)	(0000, 0000, 0100)	(0800, 0000, 0000)	(0000, 0000, 0000)	(0000, 0000, 0000)	(0000, 0000, 0000)	(0800, 0000, 0000)	(0000, 0001, 0100)
(0000, 0000, 0100)	(0800, 0001, 0000)	(0001, 0000, 0000)	(0000, 0000, 0000)	(0001, 0000, 0000)	(0800, 0000, 0000)	(0000, 0001, 0100)	(0000, 0000, 0000)
(0000, 0000, 0000)	(0000, 0000, 0100)	(0801, 0008, 1000)	(0001, 0000, 0000)	(0000, 0000, 1000)	(0001, 0000, 0000)	(0800, 0001, 1000)	(0000, 0000, 0100)
(0400, 0100, 0000)	(0010, 0800, 1000)	(0400, 0000, 0000)	(0000, 0008, 1000)	(0400, 0000, 0000)	(0100, 0000, 1000)	(0400, 0800, 0000)	(0000, 0010, 0000)
(0000, 0000, 0000)	(0400, 0000, 0000)	(0401, 0000, 0000)	(0400, 0000, 0000)	(0000, 0008, 0000)	(0400, 0000, 0000)	(0400, 0000, 0000)	(0400, 0000, 0000)
(0800, 0000, 0000)	(0401, 0000, 0000)	(0400, 0000, 0000)	(0000, 0000, 0000)	(0400, 0000, 0000)	(0400, 0008, 0000)	(0080, 0000, 0000)	(0000, 0000, 0000)
(0000, 0000, 0000)	(0800, 0010, 0000)	(0408, 0001, 0000)	(0400, 0000, 0000)	(0000, 0000, 0000)	(0400, 0000, 0000)	(0408, 0010, 0000)	(0080, 0000, 0000)
(0000, 0010, 0000)	(0408, 0000, 0000)	(0000, 0000, 0000)	(0000, 0001, 0000)	(0000, 0000, 0000)	(0408, 0000, 0000)	(0000, 0000, 0000)	(0000, 0010, 0000)

t_1 t_2
(Shape 1 (Bench) , (1000, 0101))
(Shape 2 (Boat) , (0011))

Angle	Angle Code (Hex)
0	0000 (0)
+180	0001 (1)
+90	0010 (2)
-90	0100 (4)
-180	1000 (8)

Figure 5-28: Result of the sticker compiler for 8×8 shapes, a bench and a boat. The actuator codes of the executable sticker object is hex. The hex actuator codes of the actuator model are in ( ).

```
Sticker Linker
1. For each alphabet (bit) of a given executable sticker object,
    (a) Construct the id of the alphabet.
    (b) Given the sticker place design, find a sticker place with the id.
    (c) Copy the sticker place to the executable sticker design.
2. Output the executable sticker design.
```

Figure 5-29: Algorithmic overview of sticker linker.

5.3.3 Sticker Linker

Given an executable sticker object and a sticker place design, the sticker linker generates an executable sticker design. Figure 5-29 shows an overview of the sticker linker algorithm.

Each alphabet (bit) of an executable sticker object has an id (sid, a position l, an alphabet a) (Sec. 5.3.1) each polygon of given a sticker place design also has an id (sid, a position l, an alphabet a) (Def. 5.3.5). The sticker linker generates the executable sticker design by selecting the polygons. For each bit, if the id of the bit and the id of a polygon are the same, the linker selects the polygon.

Figure 5-30 shows an example for how the sticker linker generates an executable sticker design for two shapes, the nose of a boat and a triangle. The executable sticker object (1) is generated by the sticker compiler (Fig. 5-21, 5-20). The sticker place design (2) is for a 2×2 3-2-1 sticker controller (Fig. 5-10). The executable sticker design (3) is the result of the sticker linker.

In Figure 5-30, (a) and (b) are two bits of the executable sticker object. (c) and (d) are four polygons (two red polygons and two blue polygons) (Sec. 5.3.1). (e) and (f) are two polygons of the executable sticker design. Because the id of (a) and the id of the blue polygon of (c) are the same, the sticker linker copies polygon (c) to the executable sticker design (3)(e). Because the id of (b) and the id of the red polygon of (d) are the same, the sticker linker copies the polygon (d) to the executable sticker design (3)(f). For each bit of the executable sticker object, the sticker linker copies matched polygons to the executable

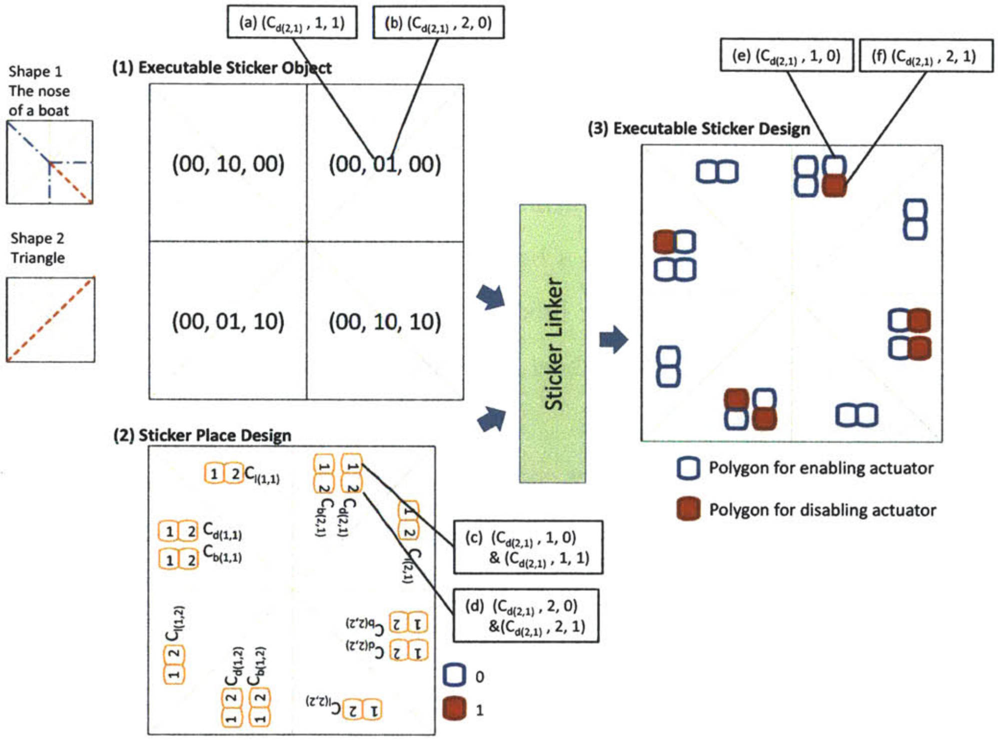


Figure 5-30: Linking an executable sticker design for the nose of boat and triangle(Fig. 5-29)

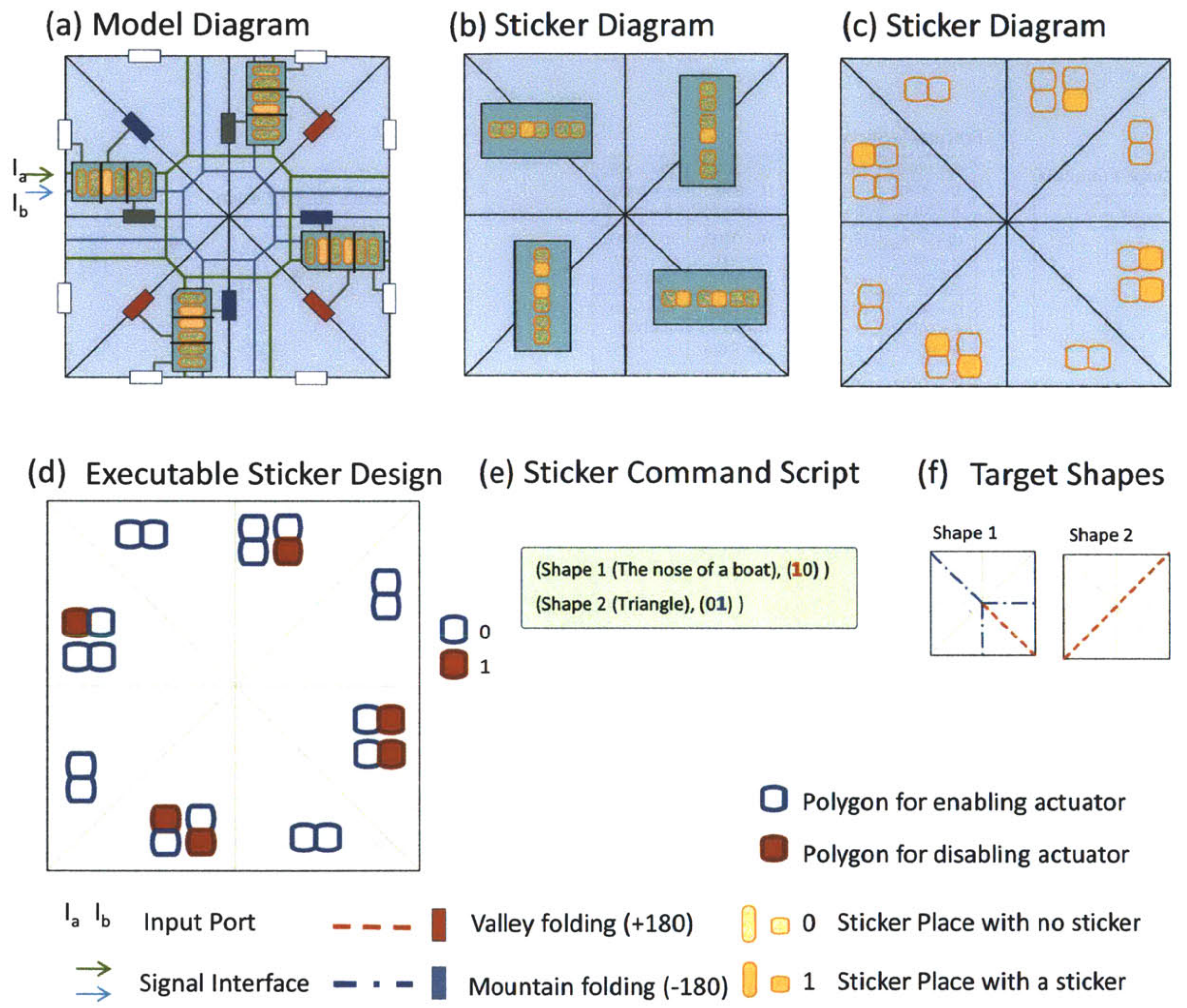
sticker design.

Figure 5-31(a)(b)(c) shows a model for 2 × 2 3-2-1 sticker controller with the example executable sticker design. (d) is the example executable sticker design for the nose of a boat and a triangle. (e) is the sticker command script for the nose of a boat and a triangle. When signal (10) is input, the model in (a) folds itself into the node of a boat. When signal (01) is input, the model in (a) folds itself into a triangle. The sticker linker generates the correct executable sticker design for target shapes.

Like the sticker compiler, the sticker linker works for various origamis. Figure 5-32 shows a result of the sticker linker for a single origami shape, an airplane. The sticker linker achieves to generate the executable sticker design for the single shape. Figure 5-33 shows a result of the sticker compiler for two shapes, an airplane and a boat while Figure 5-34 shows a result of the sticker compiler for three shapes, a boat, a tray, and a table.

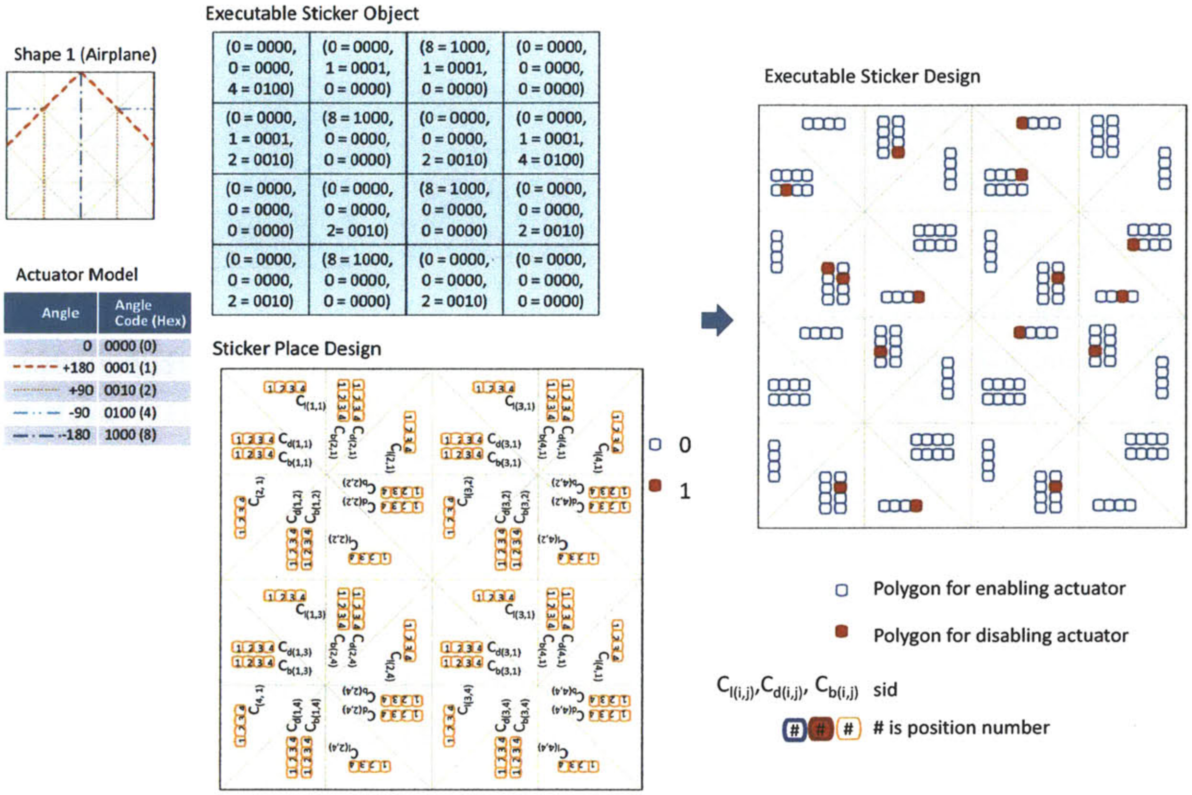
Like the sticker compiler, the sticker linker is scalable. Figure 5-35 shows a result of the sticker compiler for a 8 × 8 self-folding sheet. The sticker controller of the sheet is composed of 64 sticker controller units.

Executable sticker designs and sticker command scripts are the final results of the sticker programming algorithm. The sticker programming algorithm achieve to construct these final results.



```
(Shape 1 (The nose of a boat), (10) )
(Shape 2 (Triangle), (01) )
```

Figure 5-31: Example of a sticker place design for a 2 × 2 sticker controller for the nose of a boat and a triangle. (a)(b)(c) present a same model for the 2 × 2 sticker controller in three different diagrams. (d) presents the executable sticker design. The model contains the executable sticker according to this design. (e) is the sticker command script for the shapes. (f) is the crease patterns of the target shapes.



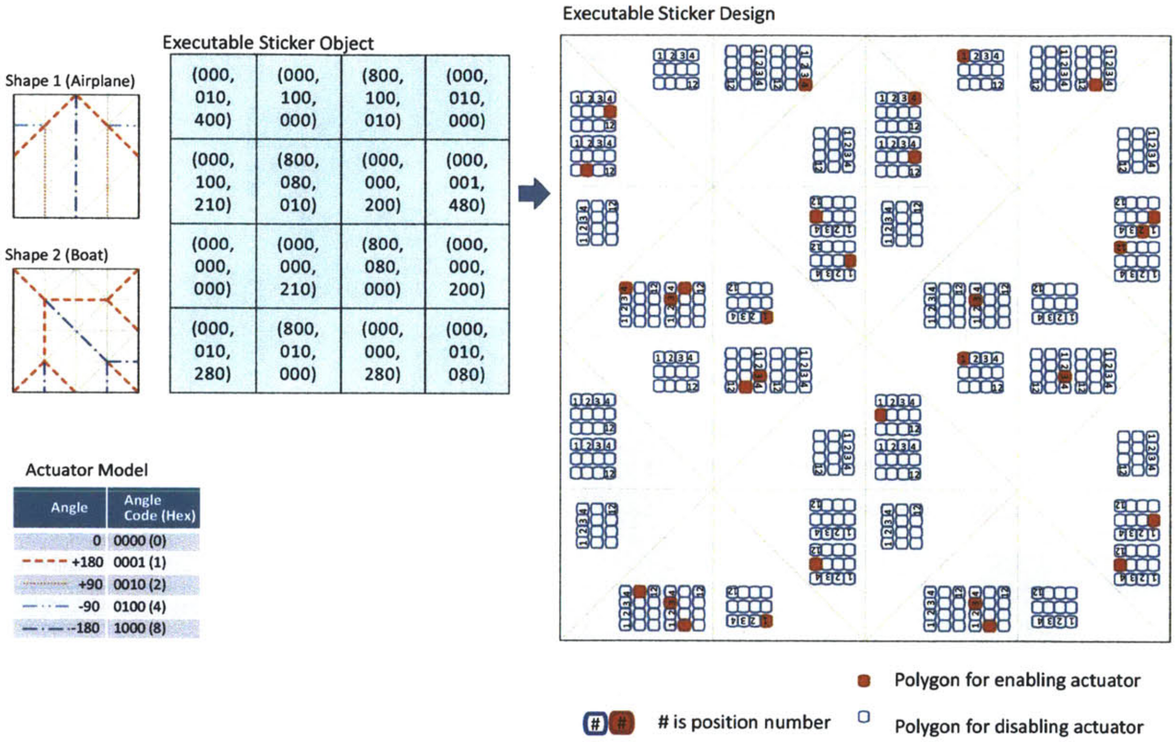
Actuator Model

Angle	Angle Code (Hex)
0	0000 (0)
+180	0001 (1)
+90	0010 (2)
-90	0100 (4)
--180	1000 (8)

Executable Sticker Object

(0 = 0000, 0 = 0000, 4 = 0100)	(0 = 0000, 1 = 0001, 0 = 0000)	(8 = 1000, 1 = 0001, 0 = 0000)	(0 = 0000, 0 = 0000, 0 = 0000)
(0 = 0000, 1 = 0001, 2 = 0010)	(8 = 1000, 0 = 0000, 0 = 0000)	(0 = 0000, 0 = 0000, 2 = 0010)	(0 = 0000, 1 = 0001, 4 = 0100)
(0 = 0000, 0 = 0000, 0 = 0000)	(0 = 0000, 0 = 0000, 2= 0010)	(8 = 1000, 0 = 0000, 0 = 0000)	(0 = 0000, 0 = 0000, 2 = 0010)
(0 = 0000, 0 = 0000, 2 = 0010)	(8 = 1000, 0 = 0000, 0 = 0000)	(0 = 0000, 0 = 0000, 2 = 0010)	(0 = 0000, 0 = 0000, 0 = 0000)

Figure 5-32: Result of the sticker linker for a single origami shape, an airplane. Given the executable sticker object (Fig. 5-26) and the 4×4 3-1-4 sticker space design (Sec. 5.3.1), the sticker linker generates the executable sticker design. The actuator codes of the executable sticker object show both the hex codes and the binary codes.


Shape 1 (Airplane)
Shape 2 (Boat)
Executable Sticker Design
Polygon for enabling actuator
# is position number
Polygon for disabling actuator

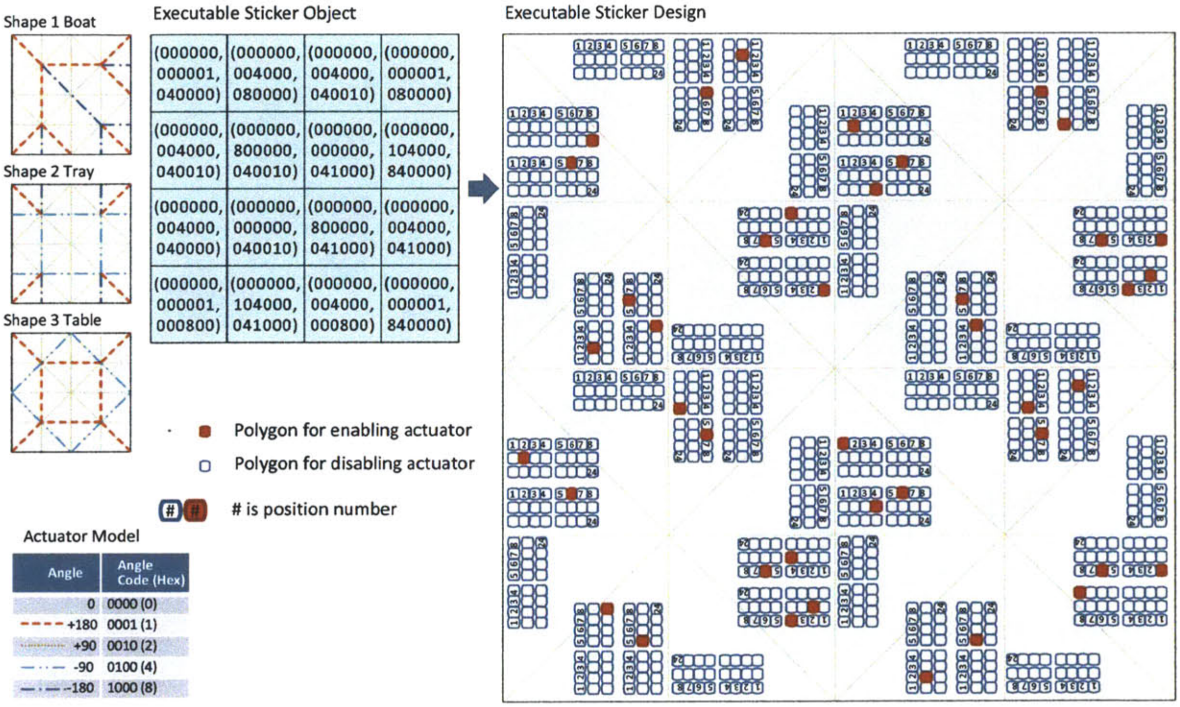
Executable Sticker Object

(000, 010, 400)	(000, 100, 000)	(800, 100, 010)	(000, 010, 000)
(000, 100, 210)	(800, 080, 010)	(000, 000, 200)	(000, 001, 480)
(000, 000, 000)	(000, 000, 210)	(800, 080, 000)	(000, 000, 200)
(000, 010, 280)	(800, 010, 000)	(000, 000, 280)	(000, 010, 080)

Actuator Model

Angle	Angle Code (Hex)
0	0000 (0)
+180	0001 (1)
+90	0010 (2)
-90	0100 (4)
-180	1000 (8)

Figure 5-33: Result of the sticker linker for two origami shape, an airplane and a boat. Given the executable sticker object (Fig. 5-21) and the 4 × 4 3-3-4 sticker space design (Sec. 5.3.1), the sticker linker generates the executable sticker design. The actuator codes of the executable sticker object are the hex codes of the actuator model. To help the reading of the executable sticker diagram, we add the position numbers in the polygons.


Shape 1 Boat
Shape 2 Tray
Shape 3 Table
Executable Sticker Design
● Polygon for enabling actuator
□ Polygon for disabling actuator
# is position number

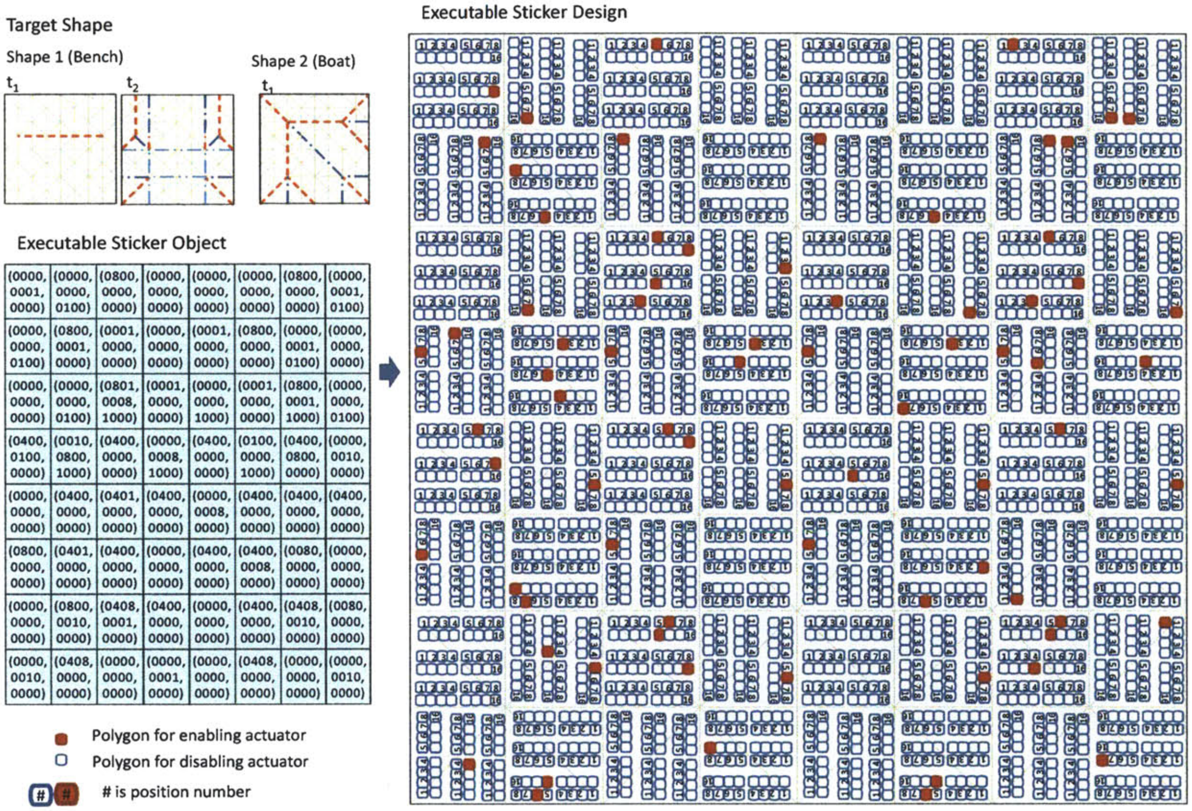
Executable Sticker Object

(000000, 000001, 040000)	(000000, 004000, 080000)	(000000, 004000, 040010)	(000000, 000001, 080000)
(000000, 004000, 040010)	(000000, 800000, 040010)	(000000, 000000, 041000)	(000000, 104000, 840000)
(000000, 004000, 040000)	(000000, 000000, 040010)	(000000, 800000, 041000)	(000000, 004000, 041000)
(000000, 000001, 000800)	(000000, 104000, 041000)	(000000, 004000, 000800)	(000000, 000001, 840000)

Actuator Model

Angle	Angle Code (Hex)
0	0000 (0)
+180	0001 (1)
+90	0010 (2)
-90	0100 (4)
-180	1000 (8)

Figure 5-34: Result of the sticker linker for three origami shape, a boat, a tray, and a table. Given the executable sticker object (Fig. 5-27) and the 4×4 3-6-4 sticker space design (Sec. 5.3.1), the sticker linker generates the executable sticker design. The actuator codes of the executable sticker object are the hex code of the actuator model. To help the reading of the executable sticker diagram, we add the position numbers in the polygons.


Target Shape
Shape 1 (Bench)
t_1
t_2
Shape 2 (Boat)
t_1
Executable Sticker Object
Polygon for enabling actuator
Polygon for disabling actuator
# is position number
Executable Sticker Design

(0000, 0001, 0000)	(0000, 0000, 0100)	(0800, 0000, 0000)	(0000, 0000, 0000)	(0000, 0000, 0000)	(0000, 0000, 0000)	(0800, 0000, 0000)	(0000, 0001, 0100)
(0000, 0000, 0100)	(0800, 0001, 0000)	(0001, 0000, 0000)	(0000, 0000, 0000)	(0001, 0000, 0000)	(0800, 0000, 0000)	(0000, 0001, 0100)	(0000, 0000, 0000)
(0000, 0000, 0000)	(0000, 0000, 0100)	(0801, 0008, 1000)	(0001, 0000, 0000)	(0000, 0000, 1000)	(0001, 0000, 0000)	(0800, 0001, 1000)	(0000, 0000, 0100)
(0400, 0100, 0000)	(0010, 0800, 1000)	(0400, 0000, 0000)	(0000, 0008, 1000)	(0400, 0000, 0000)	(0100, 0000, 1000)	(0400, 0800, 0000)	(0000, 0010, 0000)
(0000, 0000, 0000)	(0400, 0000, 0000)	(0401, 0000, 0000)	(0400, 0000, 0000)	(0000, 0008, 0000)	(0400, 0000, 0000)	(0400, 0000, 0000)	(0400, 0000, 0000)
(0800, 0000, 0000)	(0401, 0000, 0000)	(0400, 0000, 0000)	(0000, 0000, 0000)	(0400, 0000, 0000)	(0400, 0008, 0000)	(0080, 0000, 0000)	(0000, 0000, 0000)
(0000, 0000, 0000)	(0800, 0010, 0000)	(0408, 0001, 0000)	(0400, 0000, 0000)	(0000, 0000, 0000)	(0400, 0000, 0000)	(0408, 0010, 0000)	(0080, 0000, 0000)
(0000, 0010, 0000)	(0408, 0000, 0000)	(0000, 0000, 0000)	(0000, 0001, 0000)	(0000, 0000, 0000)	(0408, 0000, 0000)	(0000, 0000, 0000)	(0000, 0010, 0000)

Figure 5-35: Result of the sticker linker for 8 × 8 origami shapes, a bench and a boat. Given the executable sticker object (Fig. 5-27) and the 8×8 3-4-4 sticker space design (Sec. 5.3.1), the sticker linker generates the executable sticker design. The actuator codes in the executable sticker object are the hex codes of the actuator model. To help the reading of the executable sticker diagram, we add the position numbers in the polygons.

Chapter 6

Experiment: Self-Folding Sheets

In this chapter, we discuss an implementation (hardware and control) for self-folding sheets, and evaluate their ability to be programmed. We have built 4×4 and 8×8 self-folding sheets (Fig. 6-1). We selected four target shapes to be generated using two different self-folding sheets. We use straight-line folding and diagonal folding for evaluating the low level control for the self-folding sheet, and a space-shuttle-like shape and a hat-like shape for evaluating the high-level multi-shape self-folding planning algorithms.

6.1 Self-Folding Sheet

We have built the 4×4 and 8×8 self-folding sheets and executed three programs on them; Table 6.1 shows the overview of the target sheets.

Self-folding sheets are composed of four parts : a body, actuators, a controller, and a sticker program. (Fig. 6-2). The 4×4 sheet and the 8×8 sheet have the same body and actuators but different controllers which are implemented as different circuits. The controllers for both the 4×4 and the 8×8 sheets are implemented using the sticker controller architecture in Figure 6-2.

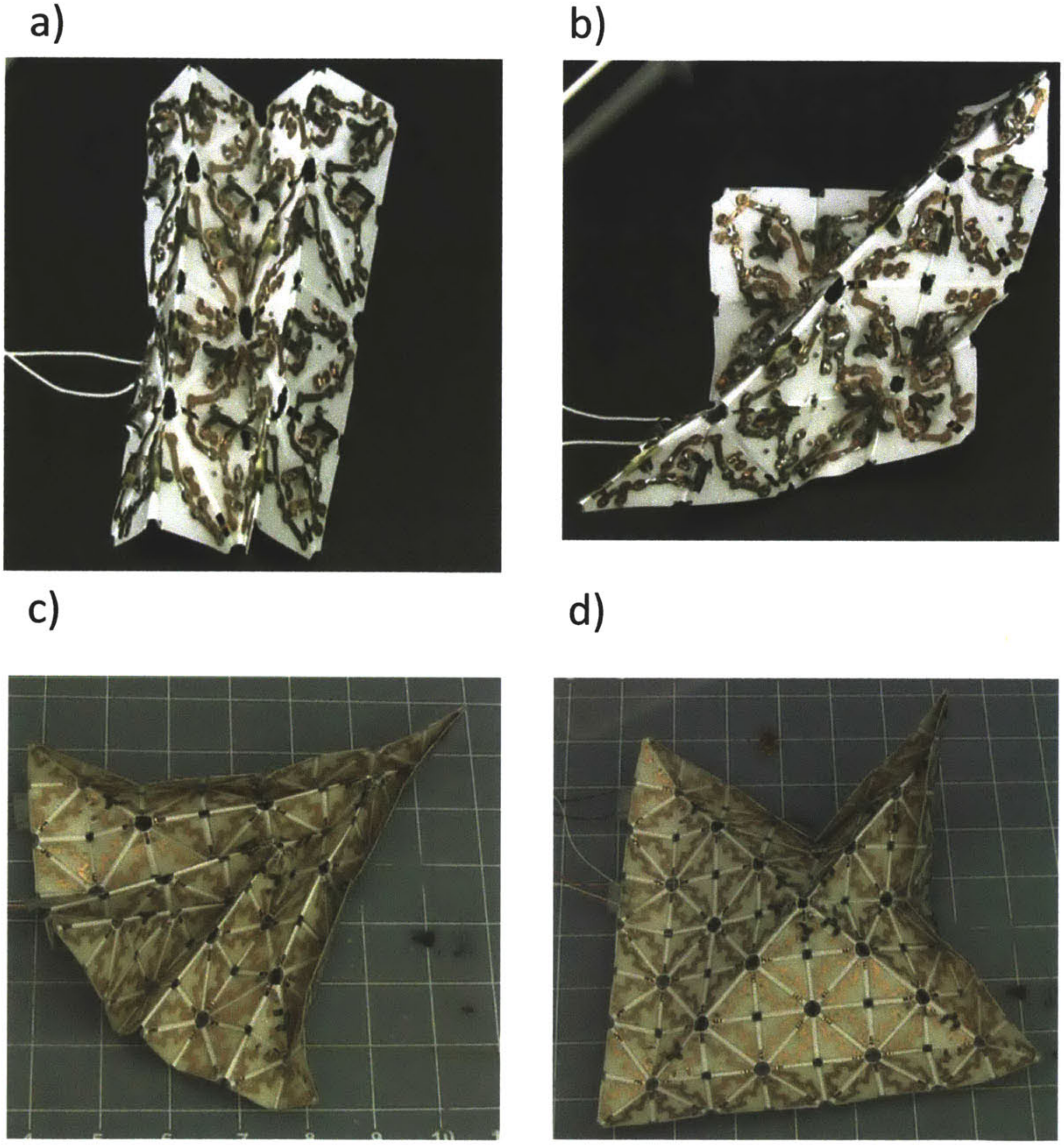


Figure 6-1: 4 Examples of self-folding sheets. (a) Vertical Folding (b) Diagonal Folding (c) Space Shuttle (d) Hat. The objects in (a) and (b) were folded from the same 4 × 4 sheet. The objects in (c) and (d) were folded from the same 8 × 8 sheet. (a)(b) 4 × 4 Self-folding sheet (c)(d) 8 × 8 Self-Folding Sheet

Table 6.1: Overview of 4 × 4 and 8 × 8 self-folding sheets

	4 × 4 sheet	8 × 8 sheet
Crease Pattern	4 × 4 Box-Pleated	8 × 8 Box-Pleated
Size	$96mm \times 96mm$	$192mm \times 192mm$
Total # of Edges	40	176
Total # of Actuators	40	36
Current	1.5 A	5.0 A
Ave. Folding Time	21.6 s	5.0 s
Sticker Controller	Sticker Controller	Socket Controller
Reprogram	Very Easy	Easy
Body	LP Body	LP Body
Actuator	Y-type Actuator	Y-type Actuator
Sticker Programming	By Sticker Programming Algorithm	By Sticker Programming Algorithm

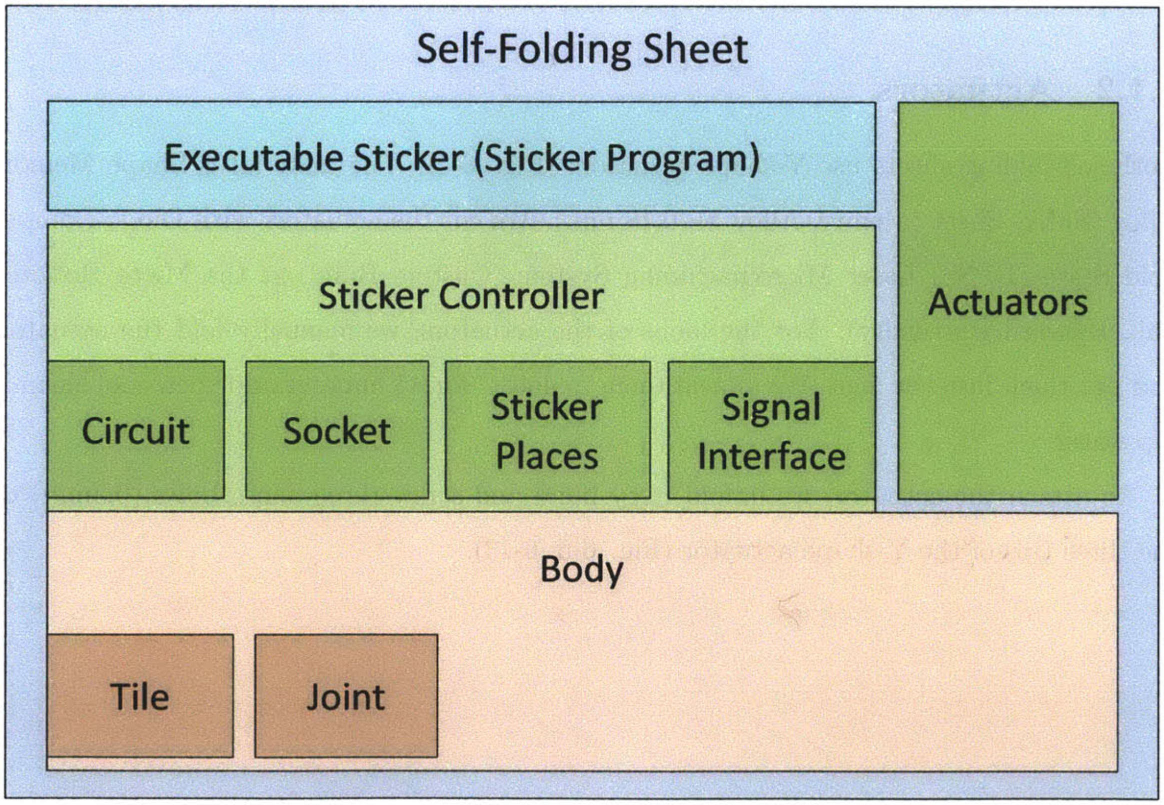


Figure 6-2: Architecture of self-folding sheet

6.1.1 The Body

The sheet body is the mechanical structure of the self-folding sheets. The body is composed of tiles and joints with box-pleated crease pattern (Fig. 6-3). Both the 4×4 and the 8×8 self-folding sheets are built from *Lamination tiles and paper joints (the LP body).* The tile is made using a stiff right-triangle material. The joint is a flexible material that connects the tiles.

The body has three layers: a lamination sheet, a paper, and a lamination sheet. The materials for the body are: lamination film (Heatseal, 0.7 mil), an anti-ageing paper (Staples, 32 lb, 649243), and micro bolts and nuts (Scale Hardware, 0.5mm). We use paper to form joints and the lamination sheet as tiles. We cut each material with the Versalaser Cuting System. To attach the three layers, we stack them and put it into a laminator (GBC, HeatSeal H425 Laminator). To align these layers, we used micro bolts and nuts (Fig. 6-3).

6.1.2 Actuators

Both self-folding sheets use Y-shape actuators (Fig. 6-4). The material is Shape Memory Alloy (SMA) Sheet (Memry, Alloy M, 0.18 mm). We cut the actuators with Diode-Pumped Sold State (DPSS) Laser Micromachining System (Custom Build, at the Micro Robotics Lab, Harvard University). For the loops of the actuators, we manually fold the actuators and put them into the gigs. We anneal them in $400°C$ for 45 minutes and then cool them in tap water.

To attach the actuator, we unfold it by hand and screw three micro bolts (0.5mm) on the three tips of the Y-shape actuator (Fig. 6-5, 6-13).

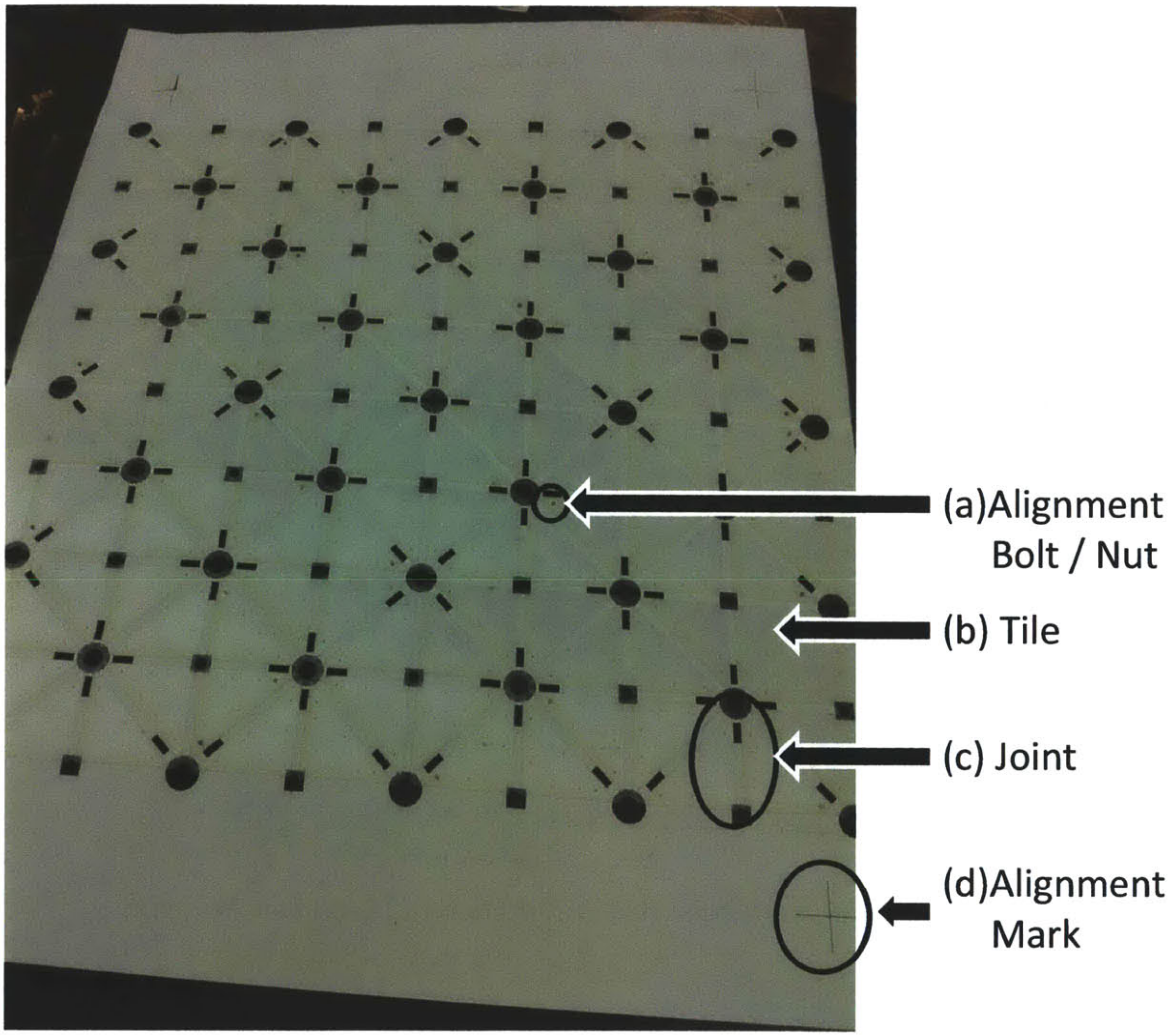


Figure 6-3: Lamination tile and paper joint body (LP Body)

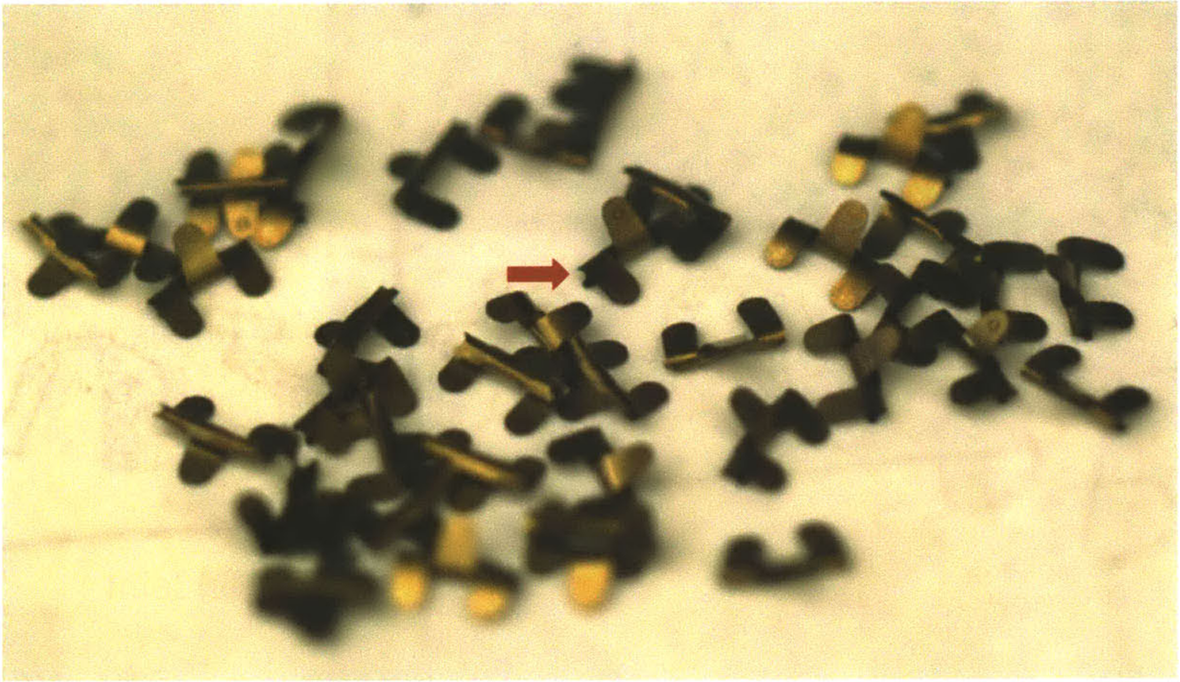


Figure 6-4: Y-Type Actuators. Red Arrow Points a Loop of an Actuator.

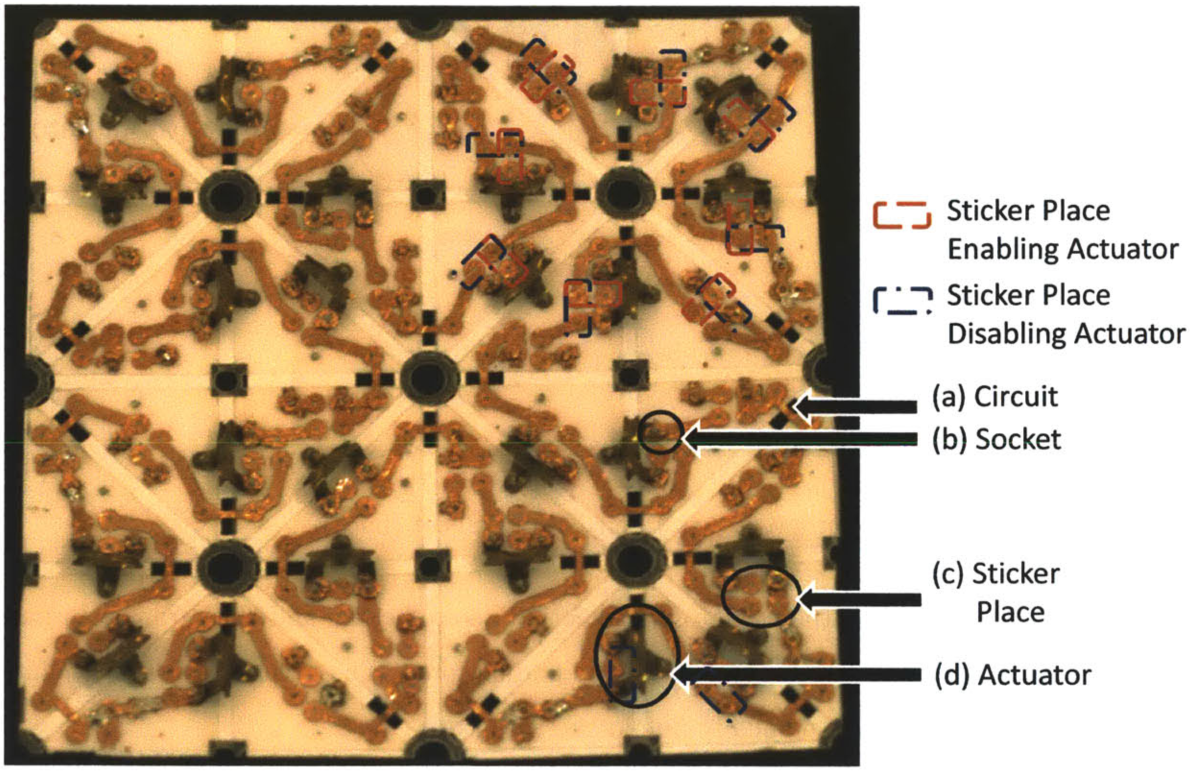

Sticker Place Enabling Actuator
Sticker Place Disabling Actuator
(a) Circuit
(b) Socket
(c) Sticker Place
(d) Actuator

Figure 6-5: 4 × 4 Self-Folding Sheet without stickers

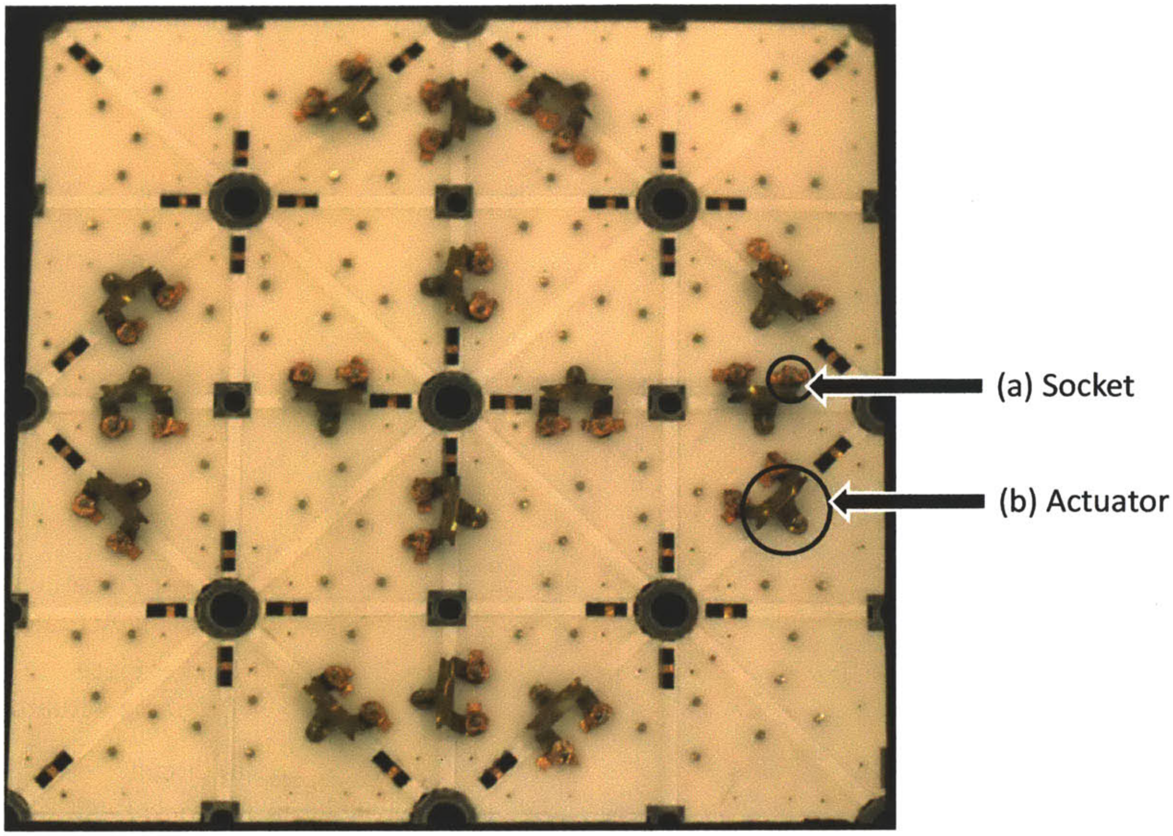


Figure 6-6: 4 × 4 Self-Folding Sheet (back)

When current passes to the actuators, the actuators are heated and they transform into the annealed shape. This motion generates the folding force. The actuators fold the joints of the self-folding sheet. We manually unfold the actuators.

The Y-shape actuator has a loop. When the actuator is exposed to heat, an annealed SMA sheet only recovers around 70% of its annealed shape. When the actuator is heated, because of its loop structure, this motion is enough to fold an edge of self-folding sheets, although the actuator cannot recover its perfect loop shape (annealed shape).

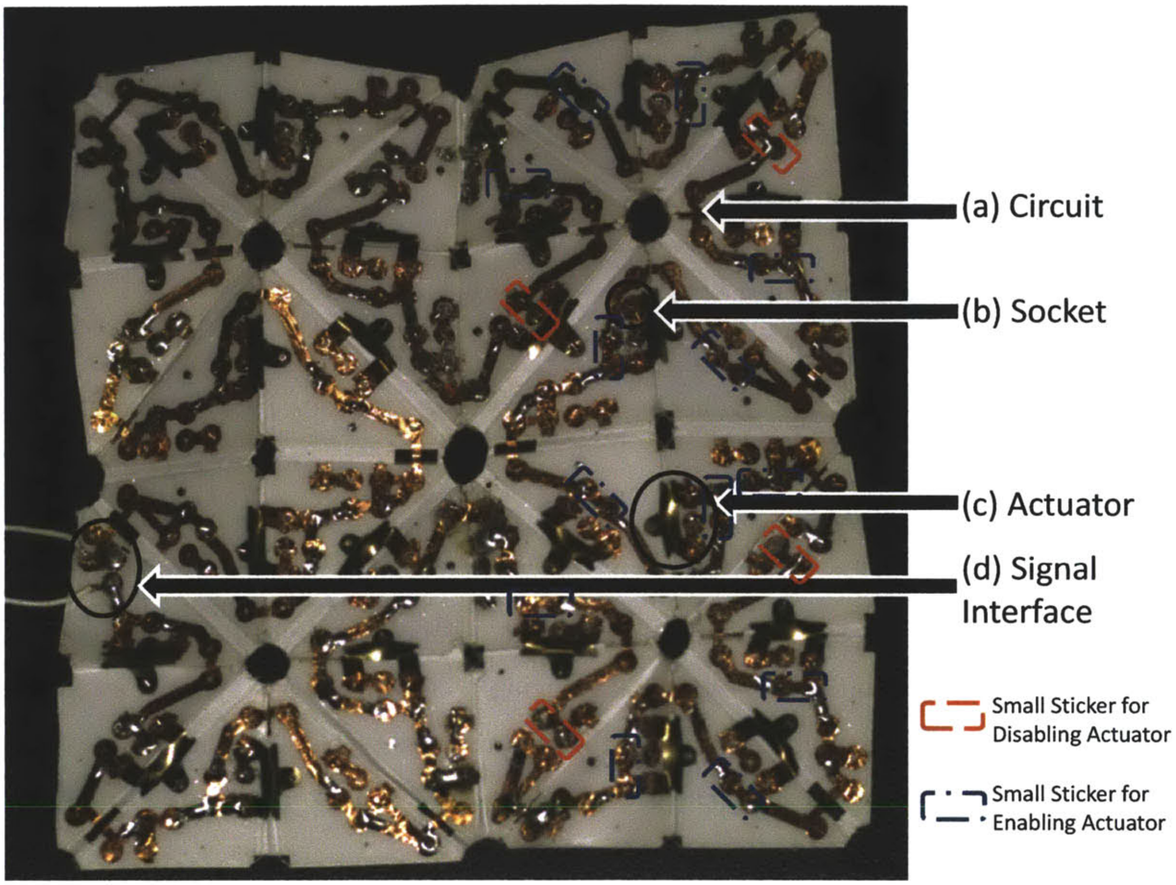

(a) Circuit
(b) Socket
(c) Actuator
(d) Signal Interface
Small Sticker for Disabling Actuator
Small Sticker for Enabling Actuator

Figure 6-7: 4 × 4 Self-Folding Sheet with the Executable Sticker (Diagonal Folding Program) (Front)

6.1.3 Sticker Controller for 4 × 4 Self-Folding Sheet

The 4 × 4 3-1-1 sticker controller is an implementation of the sticker controller architecture (Ch. 4). The sheet controller controls the actuators on the 4 × 4 self-folding sheet (Fig. 6-7).

The sticker controller is composed of a circuit, sockets, sticker places, and a signal interface (Fig. 6-5).

Circuit Scaling Algorithm (Fig. 6-12)

- Input: $i \times i$ circuit ($i \geq 2$).
- Output: $2i \times 2i$ circuit.

1. Given $i \times i$ circuit, make 3 copies of the circuit and place right, bottom, right-bottom.
2. Connect the circuits on the the center (red circle in Fig. 6-12).
3. Output $2i \times 2i$ circuit.

Figure 6-8: Constructing executable sticker object

The Circuit

The circuit (a) is a network that passes the energy for controlling and actuating.

All parts of the 4 × 4 sticker controller is on the serial circuit (Fig. 6-9). The ends (+ and -) of the serial circuit are marked on the figure. The circuit is a symmetric pattern composed of right triangles (Fig. 6-10, 6-11). The circuit is scalable with the circuit scaling algorithm. Figure 6-8 shows the circuit scaling algorithm. The 3-1-1 sticker controller has one input port of the signal interface.

Copper tape (McMaster-Carr, 76555A716) is the material used for the circuit. The copper tape is also used for the socket, and the sticker place.

The tape is cut by the DPSS Laser Micromachining System. The copper tape is composed of two layers: a copper layer and an adhesive layer. We cut the copper layer with a 20 kHz laser and then cut the adhesive layer with a 200 kHz laser. We cut the circuit on the tape and manually move it on the body (one side of the tape has adhesive).

The Socket

The sockets connect the circuit and the actuators (Fig 6-5). *Tail knot sockets* are used for the sheet controller (Fig. 6-5 and 6-13).

We used the 0.5 mm micro bolts and nuts to attach an actuator. When we attach the actuator, we make a knot on the bolt for better electronic connection.

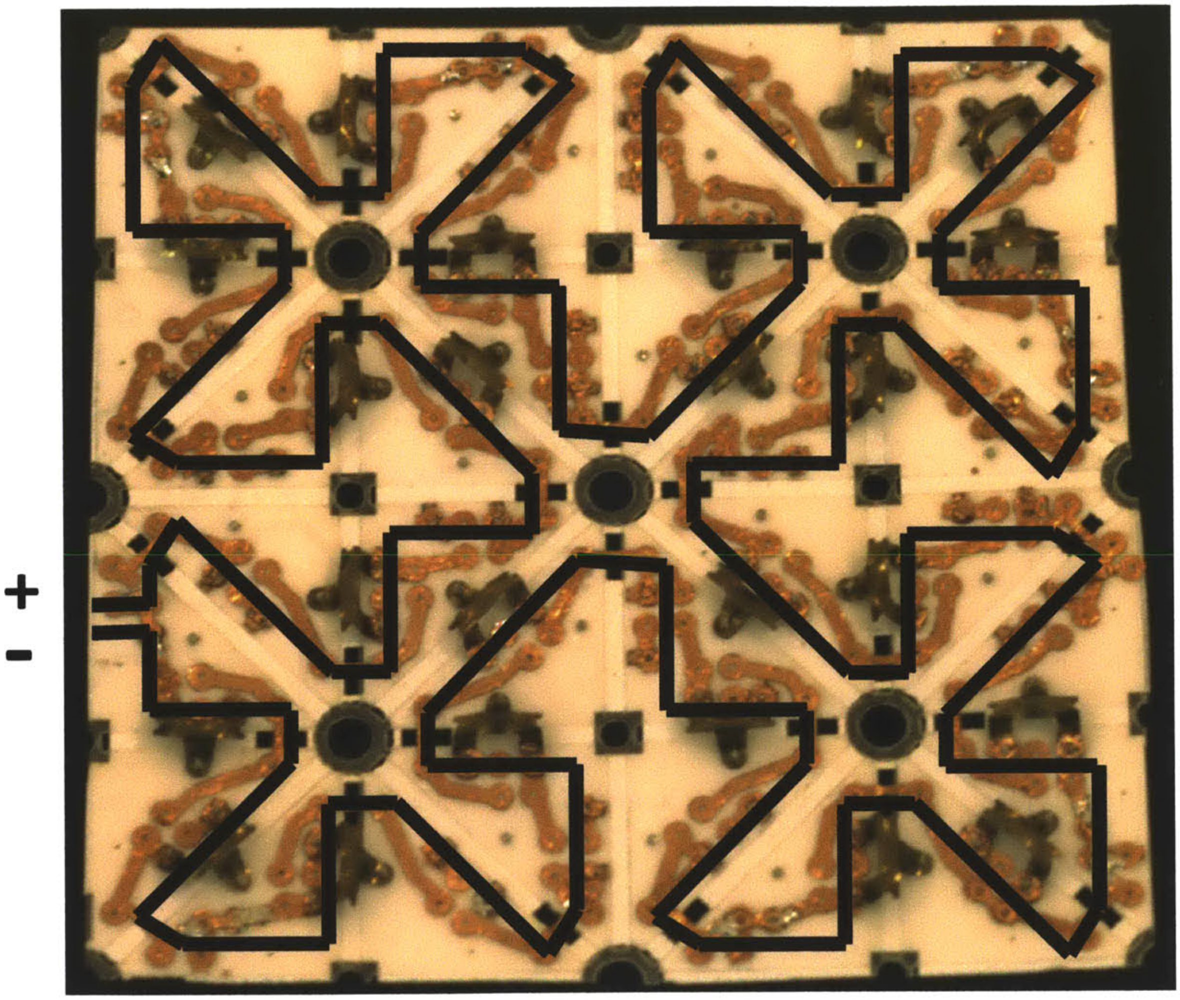


Figure 6-9: The circuit for 4 × 4 Self-Folding Sheet (with Example)

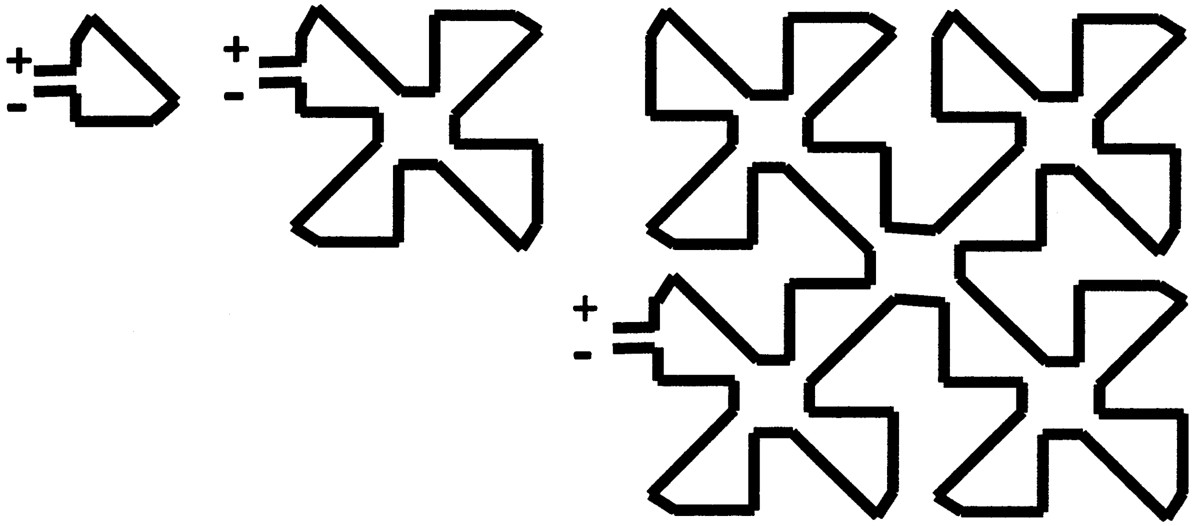


Figure 6-10: The circuits for 1 × 1, 2 × 2, and 4 × 4 Self-Folding Sheets (left, center, right)

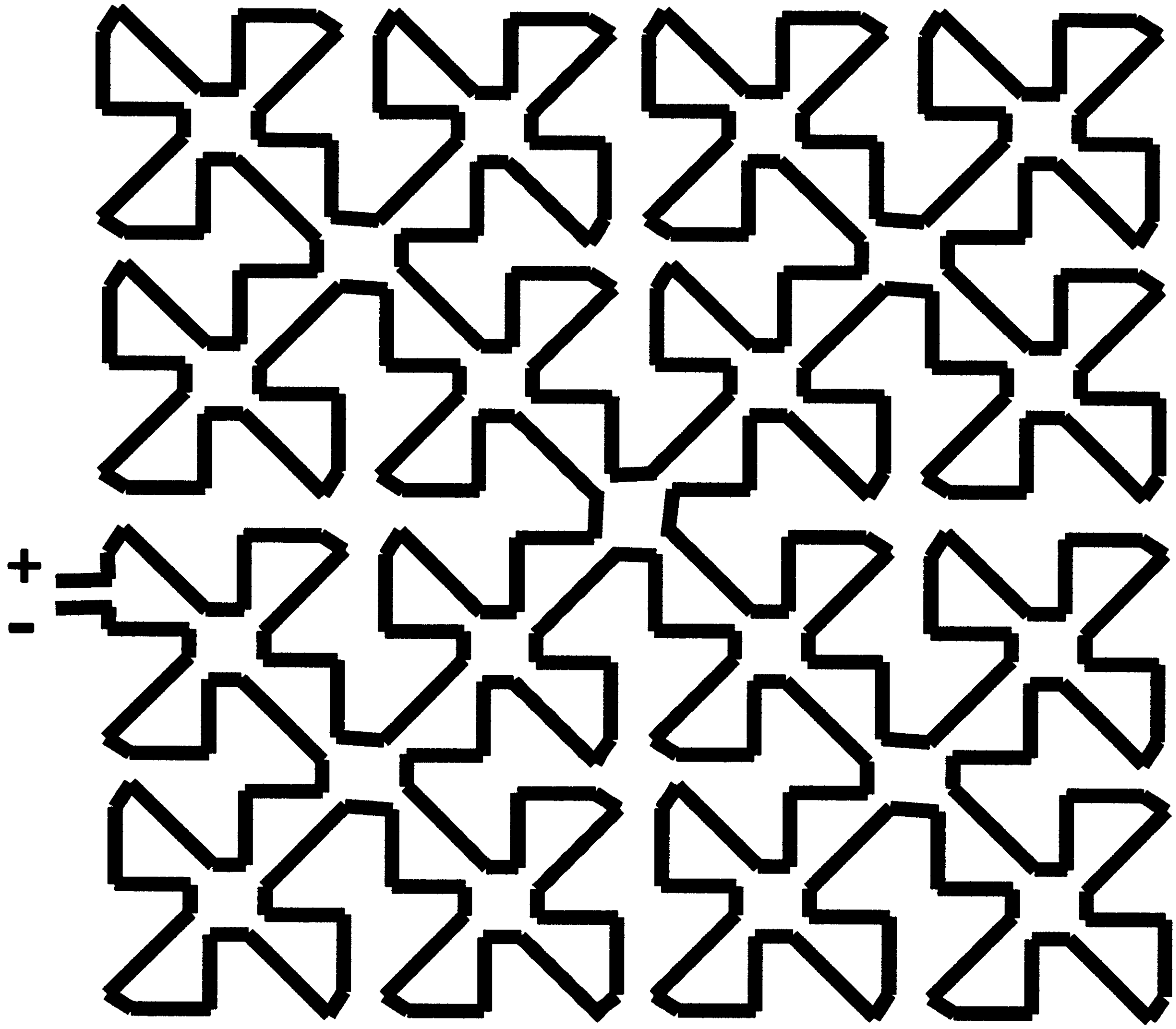


Figure 6-11: The circuit for 8 × 8 Self-Folding Sheets

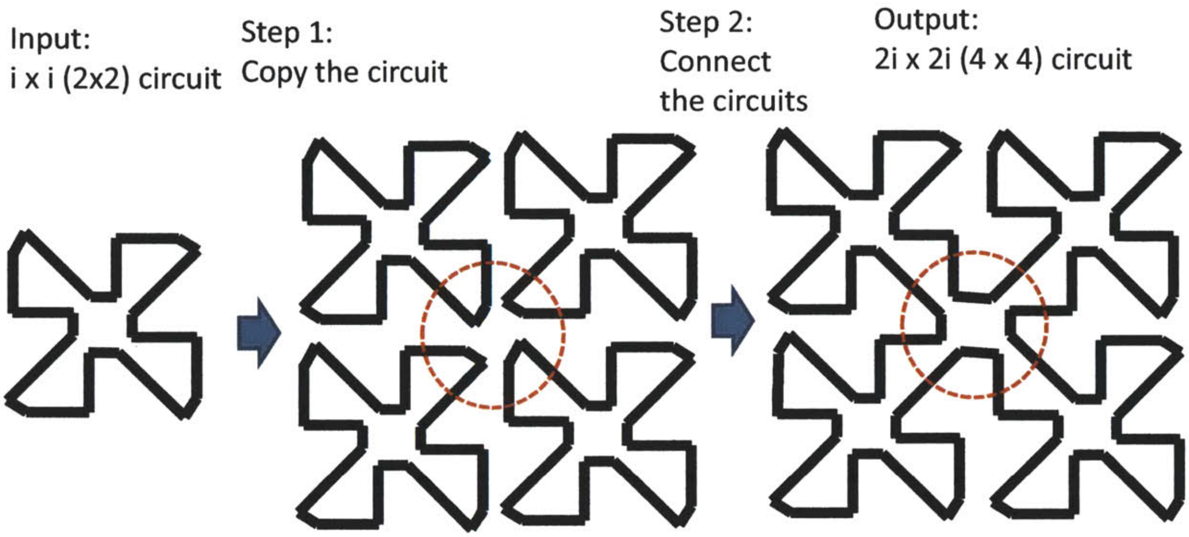


Figure 6-12: The circuit scaling algorithm

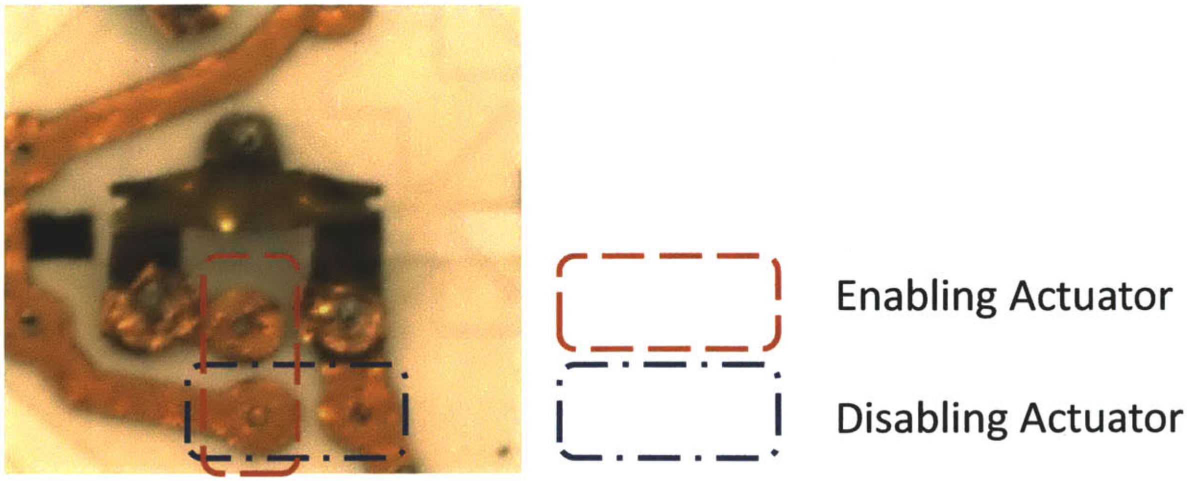


Figure 6-13: Sticker Place of 4 × 4 Self-Folding Sheet

The Sticker Place

An executable sticker is a rectangular piece of material printed according to the executable sticker designs (Ch. 5). The executable sticker for the sticker controller of the 4 × 4 sheet is composed of $2mm \times 5.6mm$ of copper tape materials. When placing a small sticker at a sticker location for enabling actuator (Fig. 6-13), current passes and activates the actuator. When placing a small sticker at a sticker location for disabling actuator (Fig. 6-13), current does not pass the actuator.

Each edge has a sticker place. The sticker place has enable and disable actuator areas (Fig. 6-13). We can add or remove stickers in different combinations. Each set of actuators in triggered by a fixed set of stickers. By replacing the stickers, we can reprogram the sticker controller.

The Signal Interface

The sticker controller receives runtime signals through a signal interface (Ch. 5). Because the sticker controller of the 4 × 4 sheet controls one actuator group, we have one input (one + and one ground) interface (Fig. 6-7).

The Executable Sticker

The executable sticker for this controller is composed of the $2mm \times 5.6mm$ patches of copper tape material (Fig. 6-14). We manually placed the stickers on the device, according to the executable sticker design for our designed motions (Fig. 6-7).

We add stickers in two steps. First, using the executable sticker design, we attach the sticker on the sticker place. Next we solder the sticker on the sticker place. After using the self-folding sheet, we can remove the stickers and reuse the device for other tasks and other shapes.

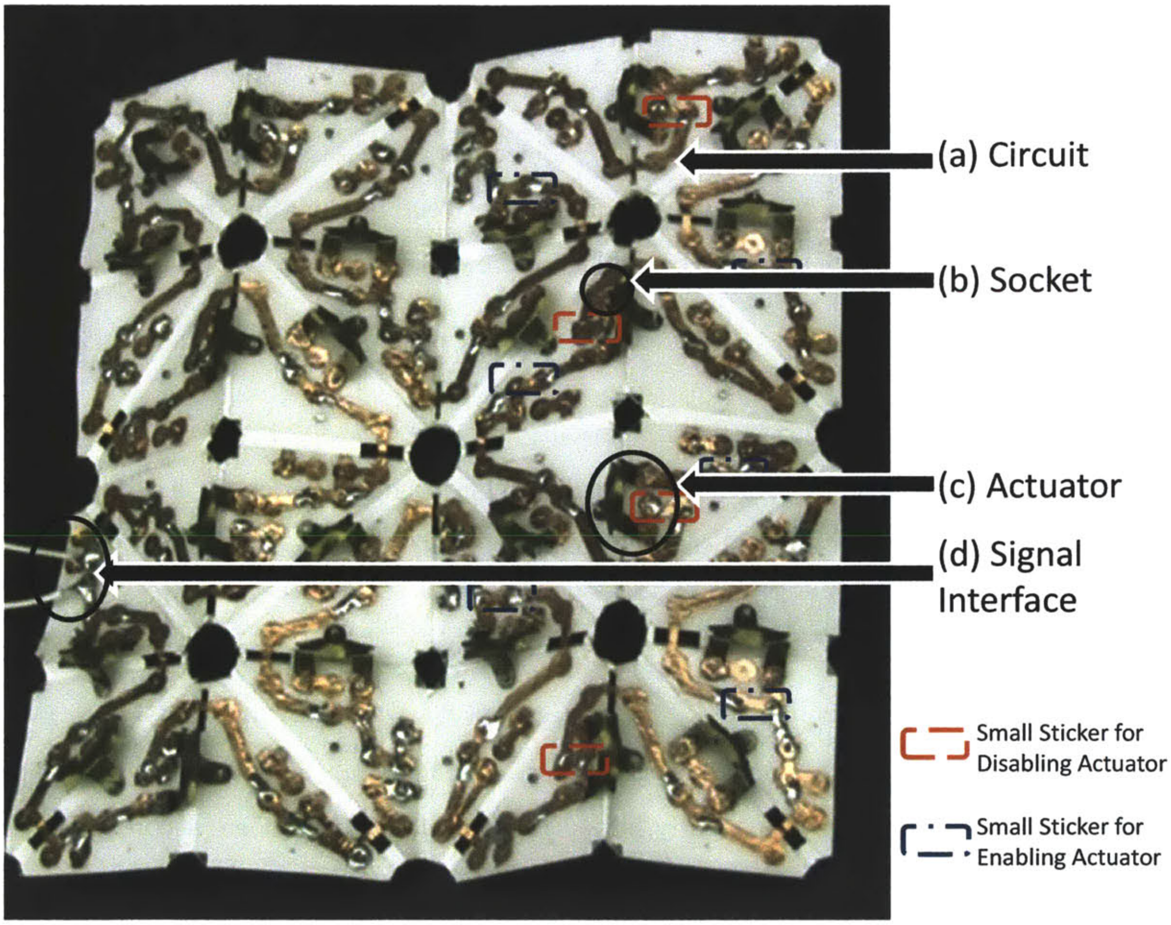

(a) Circuit
(b) Socket
(c) Actuator
(d) Signal Interface
Small Sticker for Disabling Actuator
Small Sticker for Enabling Actuator

Figure 6-14: 4 × 4 Self-Folding Sheet with Executable Sticker (Vertical Folding Program).

6.1.4 Socket Controller for 8×8 Self-Folding Sheet

A 8×8 3-3-1 socket controller is our implemented type of sticker controller (Fig. 6-15). Instead of adding or removing small stickers on sticker places, we input the control information into the controller by inserting or taking off actuators (The overall result is the same as for the 4×4 sheet: actuators are enabled to be controlled by adding or removing conductive material (i.e. stickers or actuators) to the circuit.

The Circuit

This socket controller controls three actuator groups independently. The circuit has three layers for the three actuator groups; the 3-3-1 socket controller has three input ports of the signal interface. One layer of the circuit is on the front side while the two other layers of the circuit are on the back.

Like the 4×4 sticker controller, all parts of the 8×8 sticker computer are on the serial circuit. We generate each layer of the circuit with the circuit scaling algorithm (Fig. 6-8, 6-10, 6-11). Three separated power supplies support the each layer of serial circuit.

The material of the circuit is copper foil (McMaster-Carr, 3mil, 9053K542). We place the material on the Gel-pack and cut with DPSS laser micromachine system. We manually move the circuit on an 8×8 LP body and covered by insulating covers.

PEEK with adhesive (McMaster-Carr, 2mil with adhesive, 4671T13) was the material for covering and insulating the layers of the circuit. We cut the PEEK substrate on the Versalaser Cuting System (Fig. 6-15). PEEK has an adhesive layer on the bottom.

The Hybrid Socket

The socket controller has hybrid sockets. When we input the sticker program, instead of stickers, we insert the actuators into the socket. By this programming technique, we program and control the sheet with the optimized number of actuators.

For our experiments, we populated the 40 edges of the 4×4 sheet with 40 actuators. We populated only the 36 edges relevant to our self-folding target shapes out of the 176 edges

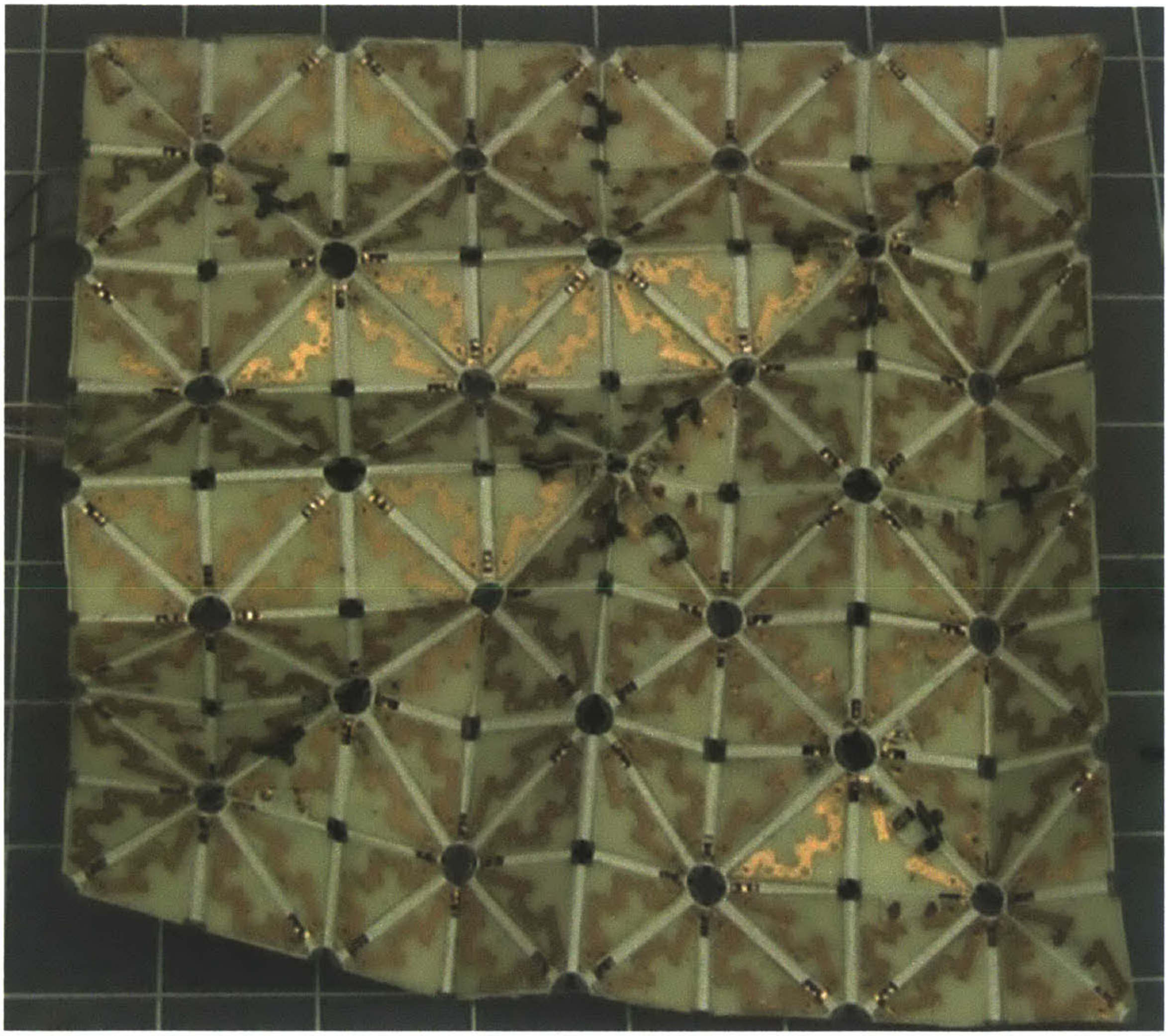


Figure 6-15: 8 × 8 Self-Folding Sheet

of the 8 × 8 sheet with actuators. The 8 × 8 self-folding sheet has 18.2% less actuators then 4 × 4 self-folding sheet, while the 8 × 8 sheets has 4.4 times more edges than the edges of the 4 × 4 sheet (Table 6.1).

Signal Interface

The signal interface has three inputs (three +s and three grounds) (Fig. 6-15(c)). We can individually send the signals to each input of the interface.

Each input that is connected each circuit layer is directly connected to the three outlets of power supplies. According to the sticker control script, we manually turn on and off the power supplies to send the signals to the socket controller.

Executable Sticker

For this socket controller, we use actuators as an executable sticker. We add an actuator in two steps. First, we remove the empty sticker from the socket. Second, we screw an actuator in the socket. If the socket was used before, there is an empty sticker. However, if the socket was never used before, two legs of the socket are connected with copper wire. Instead of removing the sticker, we disconnect this wire for the first time usage.

When we remove the actuator from the socket to change the program, we unscrew the actuator and then attach the empty sticker. Because the socket controller is a serial circuit, we need the empty sticker to keep the electronic connection to the circuit.

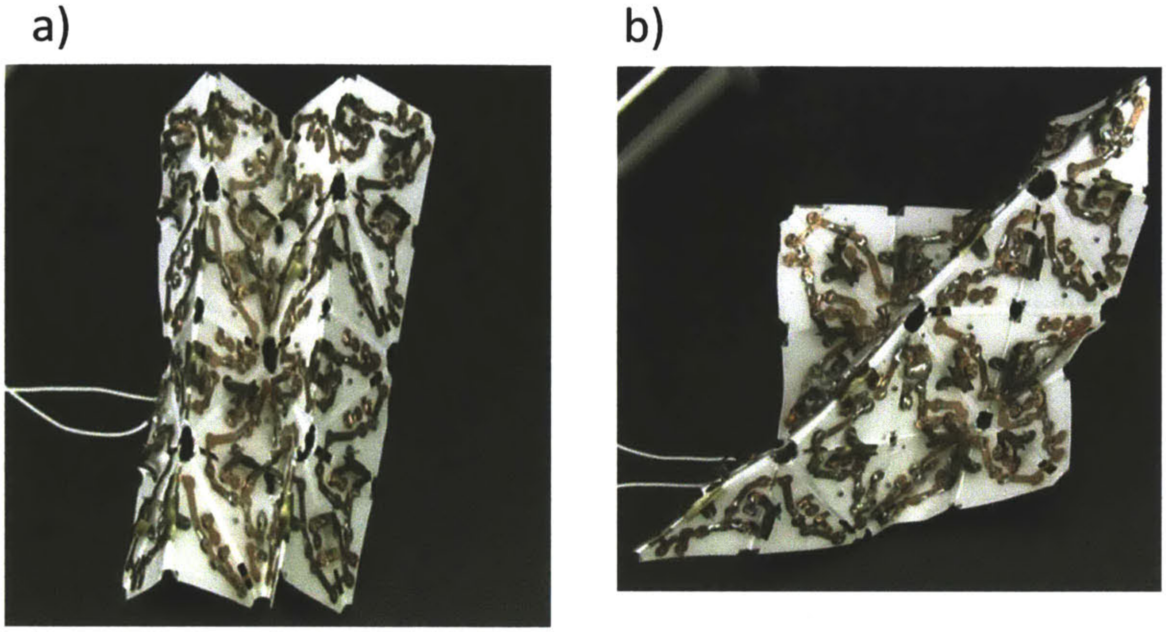


Figure 6-16: Self-Folding Sheets. (a) Vertical Folding (b) Diagonal Folding

6.2 Experiment with the 4×4 Self-Folding Sheet

The 4×4 self-folding sheet runs two basic motion: vertical and diagonal folding (Fig 6-16). Figures 6-17 and 6-18 show the vertical and diagonal folding.

We implemented and evaluated the following four steps:

1. we generated two executable sticker designs for the vertical shape and diagonal shape.
2. we placed and executed the executable sticker for the vertical folding.
3. we removed the executable sticker.
4. we placed and executed the executable sticker for the diagonal folding.

6.2.1 Sticker Programming for Vertical and Diagonal Folding

We generated two executable sticker designs for the two basic shapes with the sticker programming algorithm (Ch. 5). Figure 6-19 shows the design output. The two target shapes

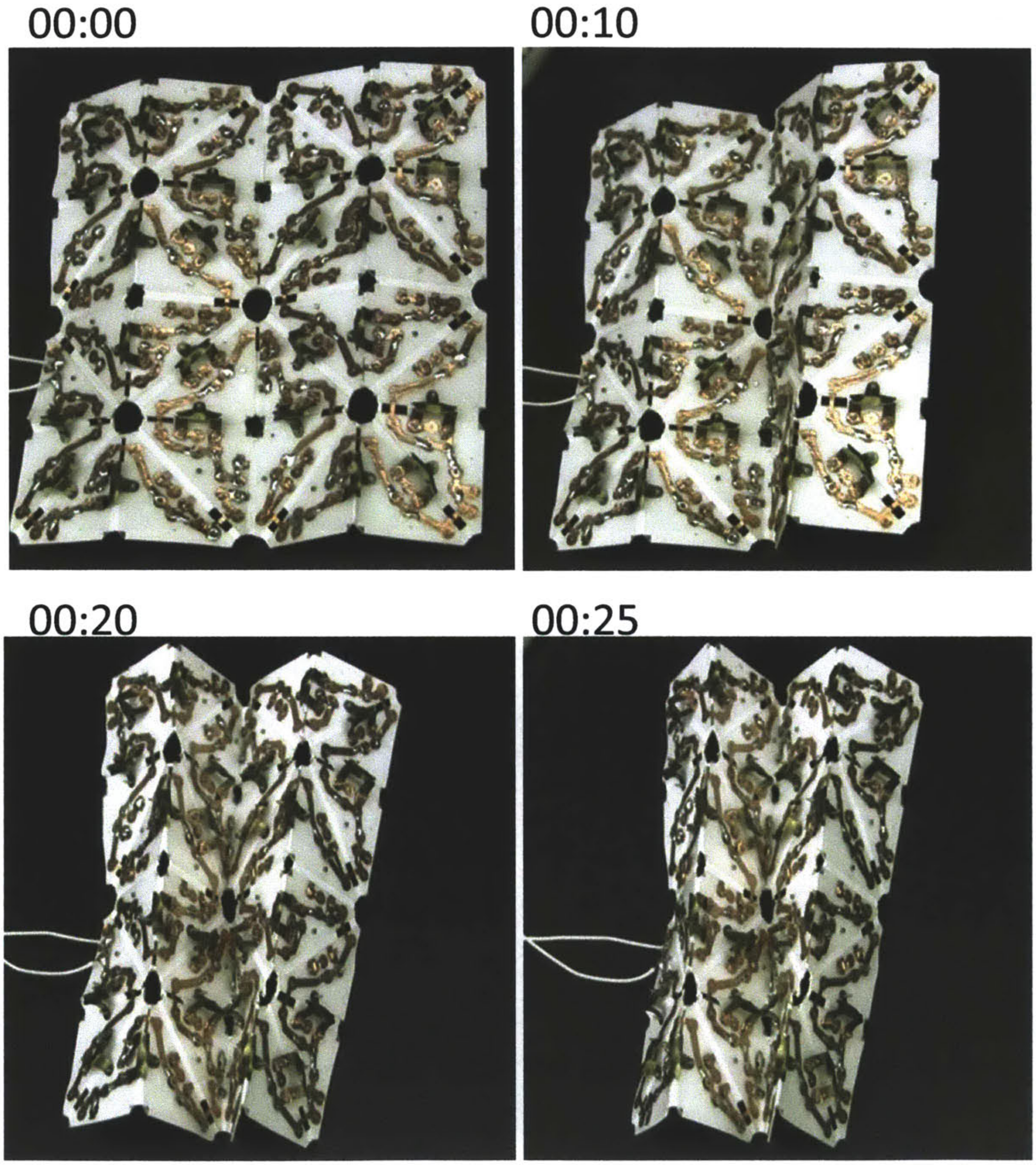


Figure 6-17: Snapshots from controlling the vertical folding 4 × 4 Self-Folding Sheet

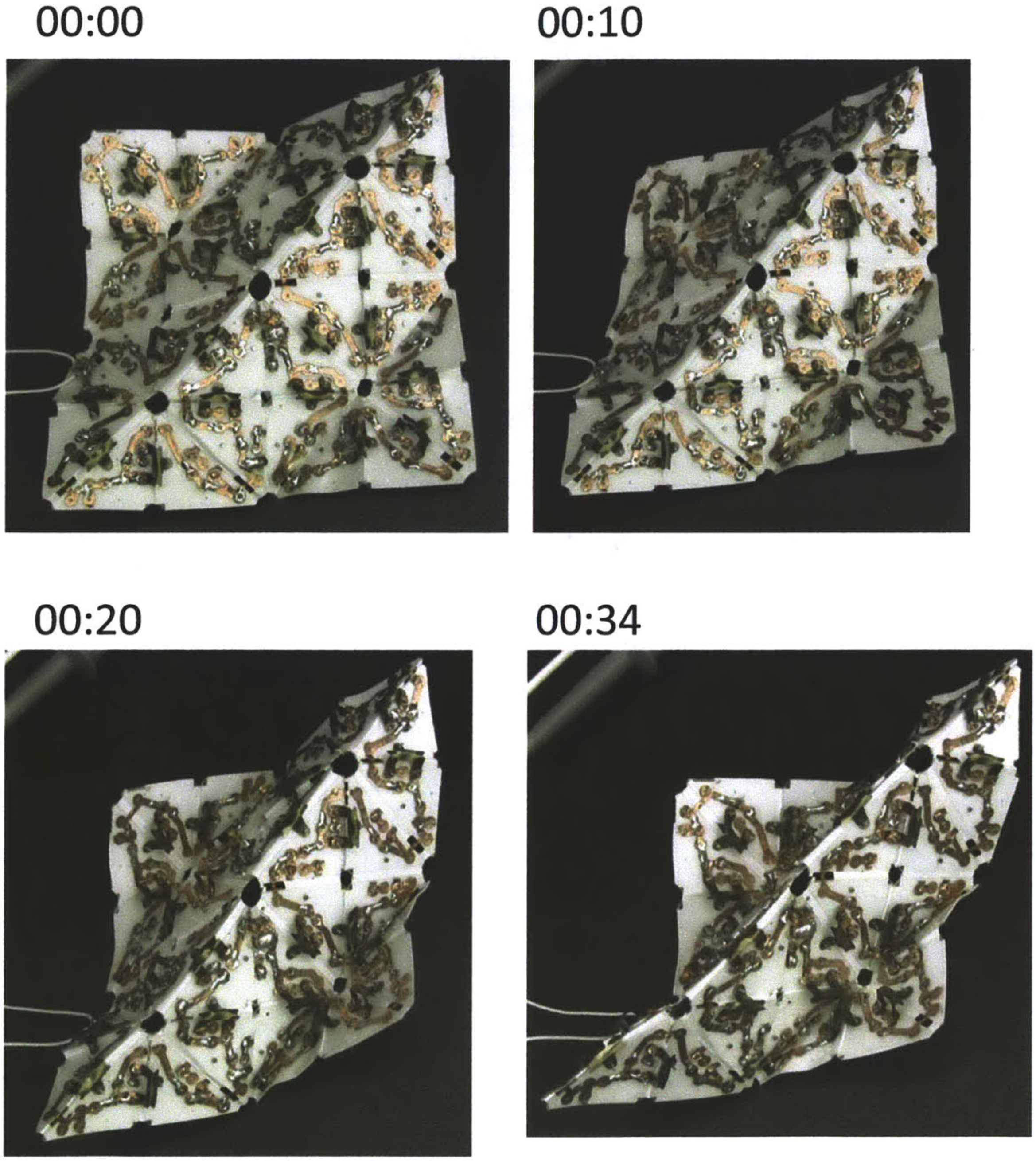


Figure 6-18: Snapshots from controlling the diagonal folding 4 × 4 Self-Folding Sheet

Table 6.2: Origami Planning Time for Vertical and Diagonal folding

Single Origami Planing Analysis Time (Vertical)	3.6 s (3600 ms)
Single Origami Planing Building Time (Vertical)	17 ms
Single Origami Planing Analysis Time (Diagonal)	4.2 s (4200 ms)
Single Origami Planing Building Time (Diagonal)	16 ms

CPU	Intel Core 2 Quad 2.83GHz (Q9550)
Storage	3 GB RAM, Seagate 750GB 300MBps 7200rpm HDD
Graphics	NVIDIA Quadro FX 1700

are inputs to the algorithm (Fig. 6-20). We automatically planned two target shapes with the origami planner [2] (Fig. 6-21, 6-22, 6-23) and then manually computed executable sticker designs the compiling algorithm (Fig. 6-19). Figures 6-21 and 6-22 show snapshot from the origami planning of vertical and diagonal folding. Table 6.2 shows the planning times on hardware.

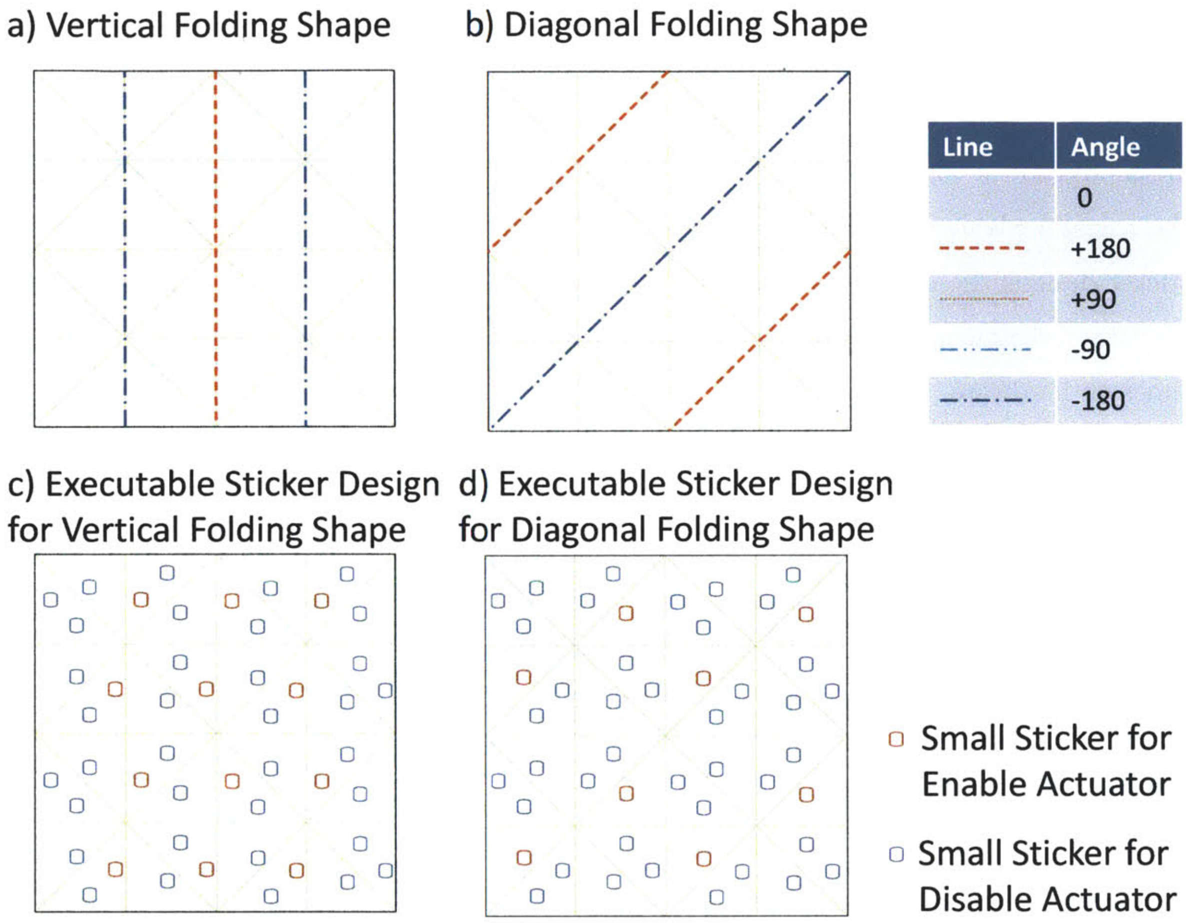

a) Vertical Folding Shape
b) Diagonal Folding Shape
Line
Angle
0
+180
+90
-90
-180
c) Executable Sticker Design for Vertical Folding Shape
d) Executable Sticker Design for Diagonal Folding Shape
Small Sticker for Enable Actuator
Small Sticker for Disable Actuator

Figure 6-19: Results of the sticker programming algorithm. (a)(b) Origami plan. (c)(d) Executable sticker design. Each small square shows the sticker type for each sticker place. We input the vertical folding program according to (c) as shown in Figure 6-7. We input the diagonal folding program according to (d) as shown in Figure 6-14.

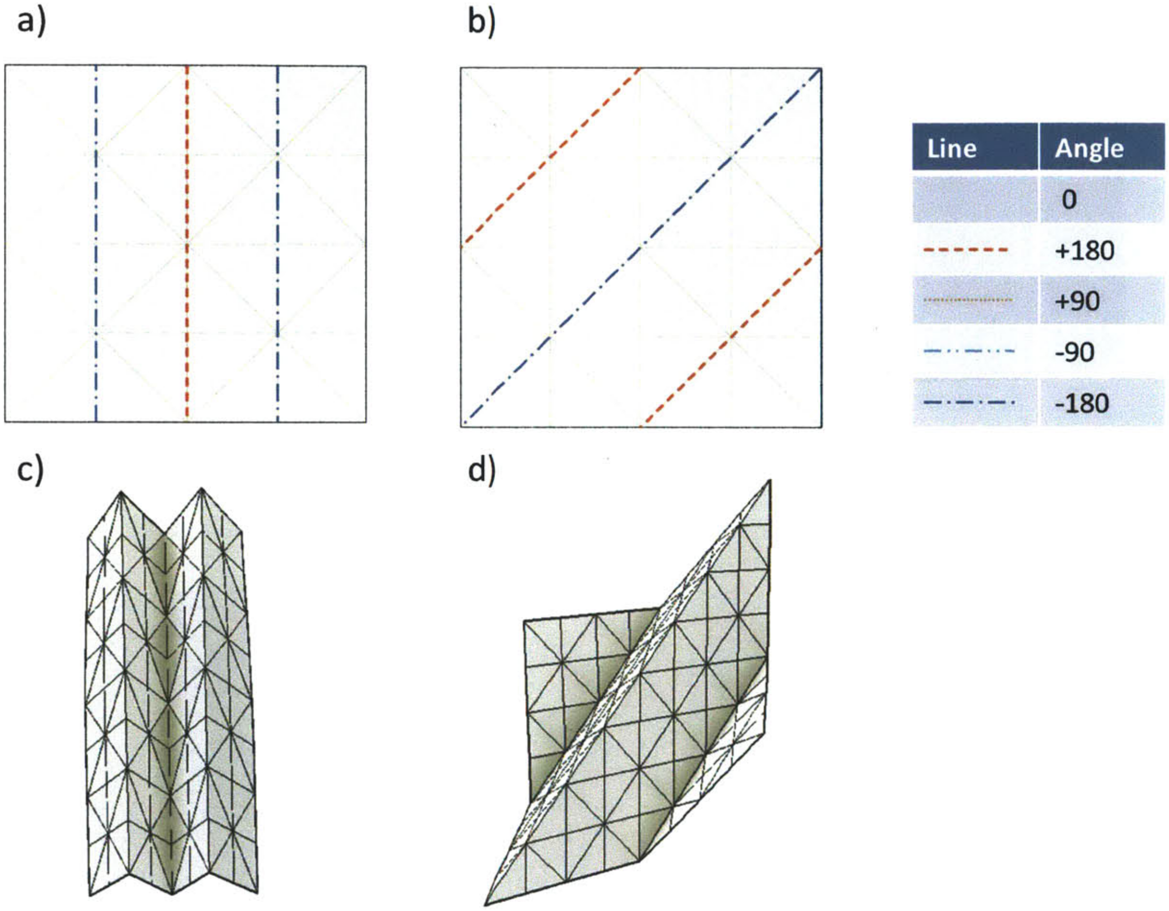


Figure 6-20: Target Shapes for Basic Motion. (a)(c) Vertical Folding (b)(d) Diagonal Folding

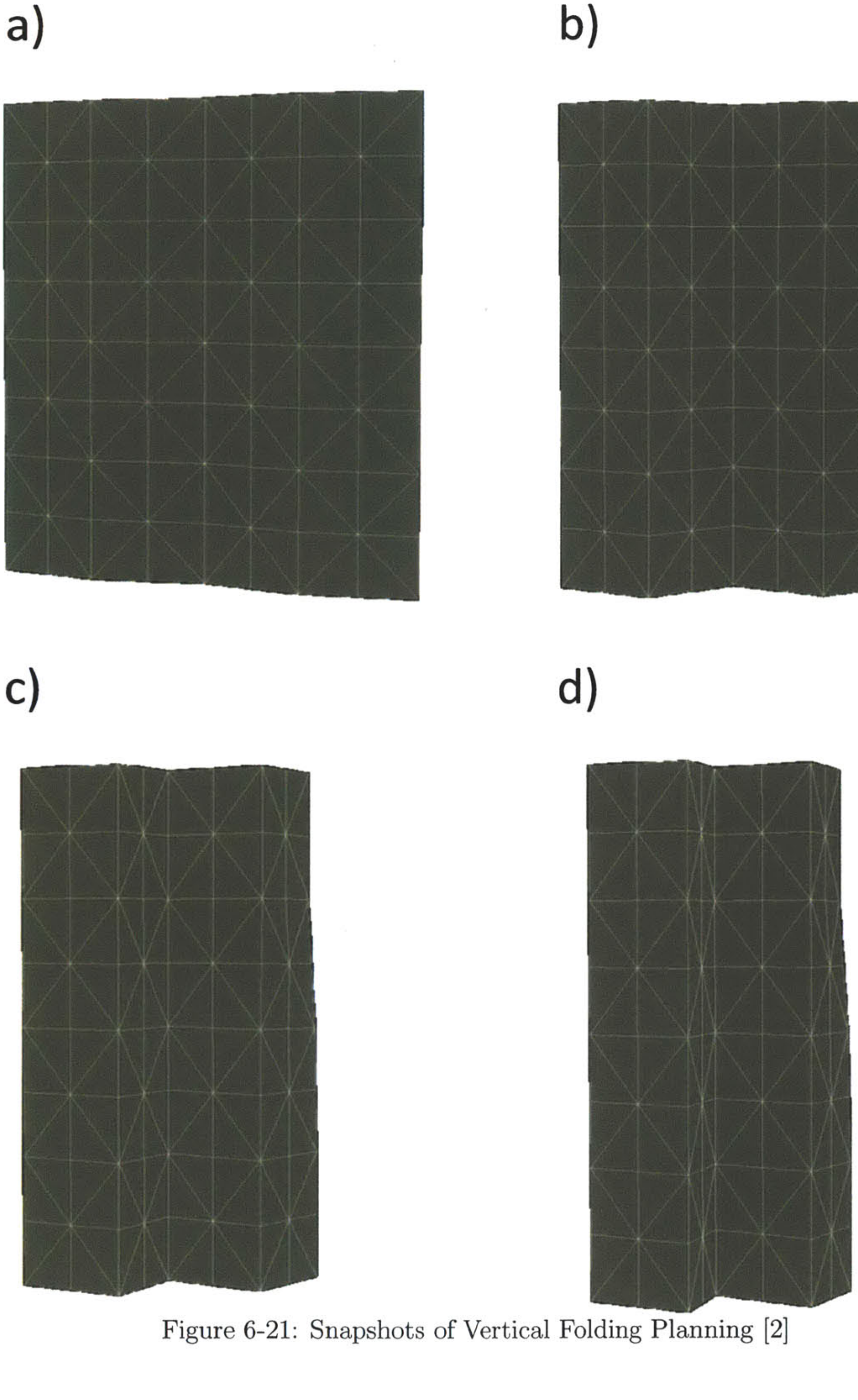


Figure 6-21: Snapshots of Vertical Folding Planning [2]

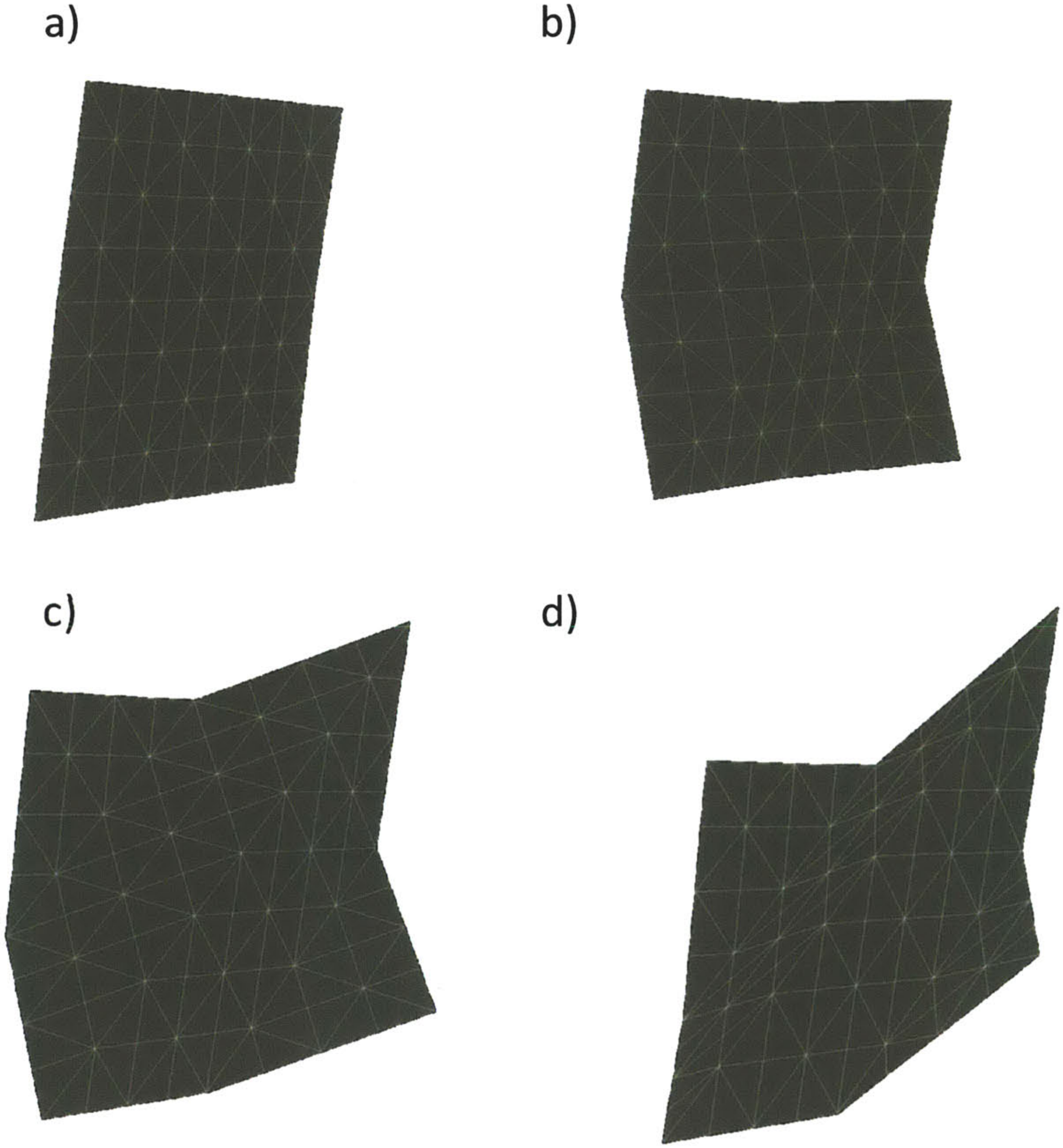


Figure 6-22: Snapshots of Diagonal Folding Planning [2]

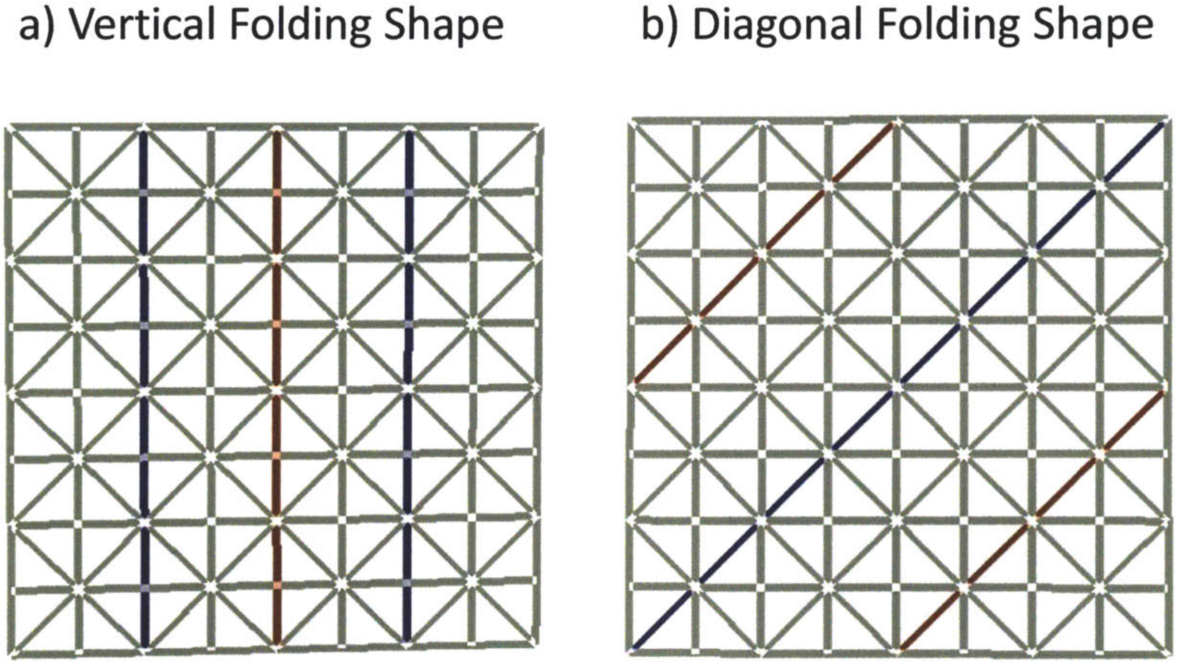


Figure 6-23: Snapshot of Sticker Program Planing. The red line denotes +180° folding. The blue line denotes -180° folding.

Table 6.3: Actuators of 4 × 4 Sheet

	Folding Actuators	Total Actuators	Total Edges	Folding Actuators / Total Actuators	Total Actuators / Total Edges
Vertical	12	40	40	30.0%	100.0%
Diagonal	10	40	40	25.0%	100.0%
Total	11	40	40	42.5%	100.0%

Table 6.4: Folding Time and Current of 4 × 4 Sheet

	# of Runs	Current	Ave. Folding Time
Vertical	14	1.5 A	21.0 s ±26.7%
Diagonal	13	1.5 A	22.4 s ±17.9%
Total	27	1.5 A	21.6 s ±22.5%

6.2.2 Results

The 4 × 4 sheet has 40 actuators and 40 edges. 42.5% of the actuators were used for each of the two shapes (Table 6.3).

First, we executed the vertical folding program on the 4 × 4 self-folding sheet 14 times. Second, we removed the program and reprogrammed the sheet diagonal folding program on the sheet. Then we executed the diagonal folding 13 times. The 4 × 4 sheet achieved the vertical and diagonal folding reliably (Fig. 6-17, 6-18). The 4 × 4 self-folding sheet runs with current set at 1.5 A. The average folding time of both shapes is 21.6 s (Table 6.4).

The average angle[1] of the basic folding motion is 134.0° ± 12.1% Our target folding angle for the basic folding motion was 180.0°. We achieved 74.5% of the target angle.

The error of the diagonal folding angle is 2.1 times bigger than the error of the vertical folding. The diagonal folding is achieved by folding three straight-lines. Each line has the same length and the same number of the actuators. But, diagonal folding is achieved by folding one long center-line and two short side-lines. Although the center-line carried more weight than the side-line, the center-line achieved better folding than on the side-line. Four actuators are on the center-line while two actuators on each side-line.

[1] The angles might not be accurate. We measured the angles by video analysis after the experiments. We picked and analyzed three angles from the first videos of each experiment.

Table 6.5: Folding Angle and Folding Achievement of 4 × 4 Sheet

	Ave. Folding	Target	Folding Achievement
	Angles [1]	Angles	(Folding Angle / Target Angle)
Vertical	141.6° ± 7.9%	180.0°	78.7%
Diagonal	126.4° ± 16.3%	180.0°	70.2%
Total	134.0° ± 12.1%	180.0°	74.5%

Table 6.6: Failure of 4 × 4 Sheet

	# of Runs	# of Failure	Ave. Failure
Vertical	14	1 (of 14 runs)	0.7 (of 10 runs)
Diagonal	13	2 (of 13 runs)	1.5 (of 10 runs)
Total	27	3 (of 27 runs)	1.1 (of 10 runs)

While we folded the 4 × 4 sheet 27 times, the experiment failed to meet the goal three times (Table 6.6). Most of failures were due to broken or weak connection between the socket and the actuator. SMA, a material used, is hard to solder. We made the electronic connection not only with solder but also with conductive bolts and nuts. However, while the sheet folded several times, the electronic connection was weak. Once the connection was loose, the socket was hard to recover. In this case, we fixed the system by disabling the broken actuator.

The average number of disabled actuators was 1.04 (Table 6.7). The sheet achieved its goal shapes reliably despite the number of the disabled actuators.

Most of the results of the two basic shapes on the 4 × 4 sheet are similar. However, the resistance was 19.1 Ω for vertical folding while the resistance was 28.9 Ω for diagonal folding. The resistance of the sheet increased 1.5 times after we reprogrammed the sheet. (Table 6.8). Because the number of folding actuators is almost same in the two experiments, we

Table 6.7: Disabled Actuators of 4 × 4 Sheet

	Ave. # of	Folding	Disabled Actuators	Disabled Actuators
	Disabled Actuators	Actuators	/ Folding Actuators	/ Total Actuators
Vertical	0.77	12	6.4 %	1.9 %
Diagonal	1.36	10	13.6 %	3.4 %
Total	1.04	11	9.7 %	2.6 %

Table 6.8: Resistance of 4 × 4 Sheet

	Ave. Resistance (R)
Vertical	19.1 Ω
Diagonal	28.9 Ω
Total	23.6 Ω

can say the connectivity decreases after reprogramming the sheet.

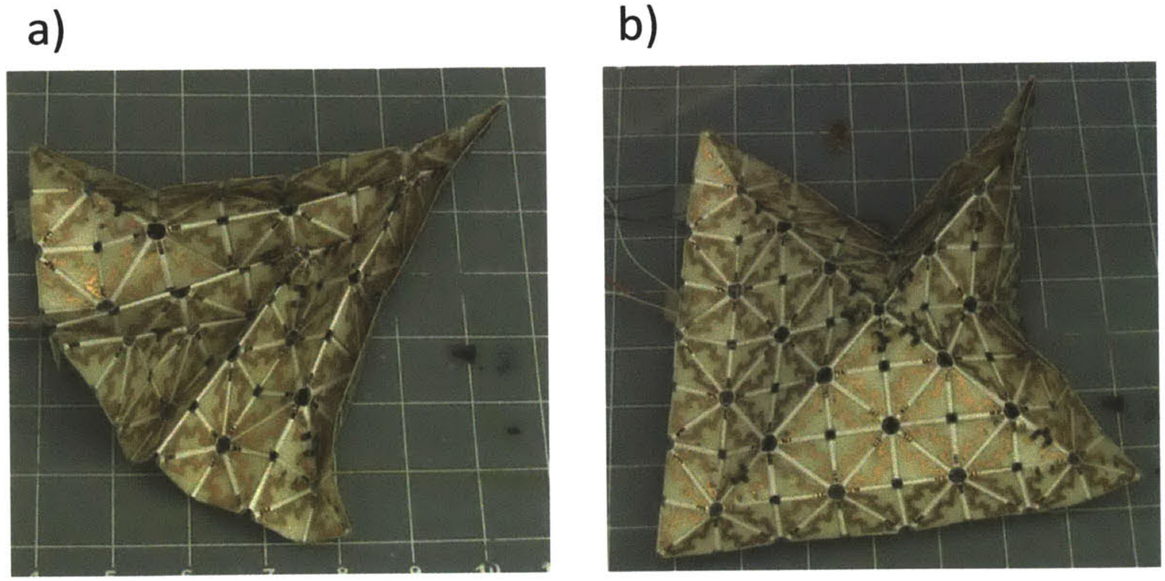


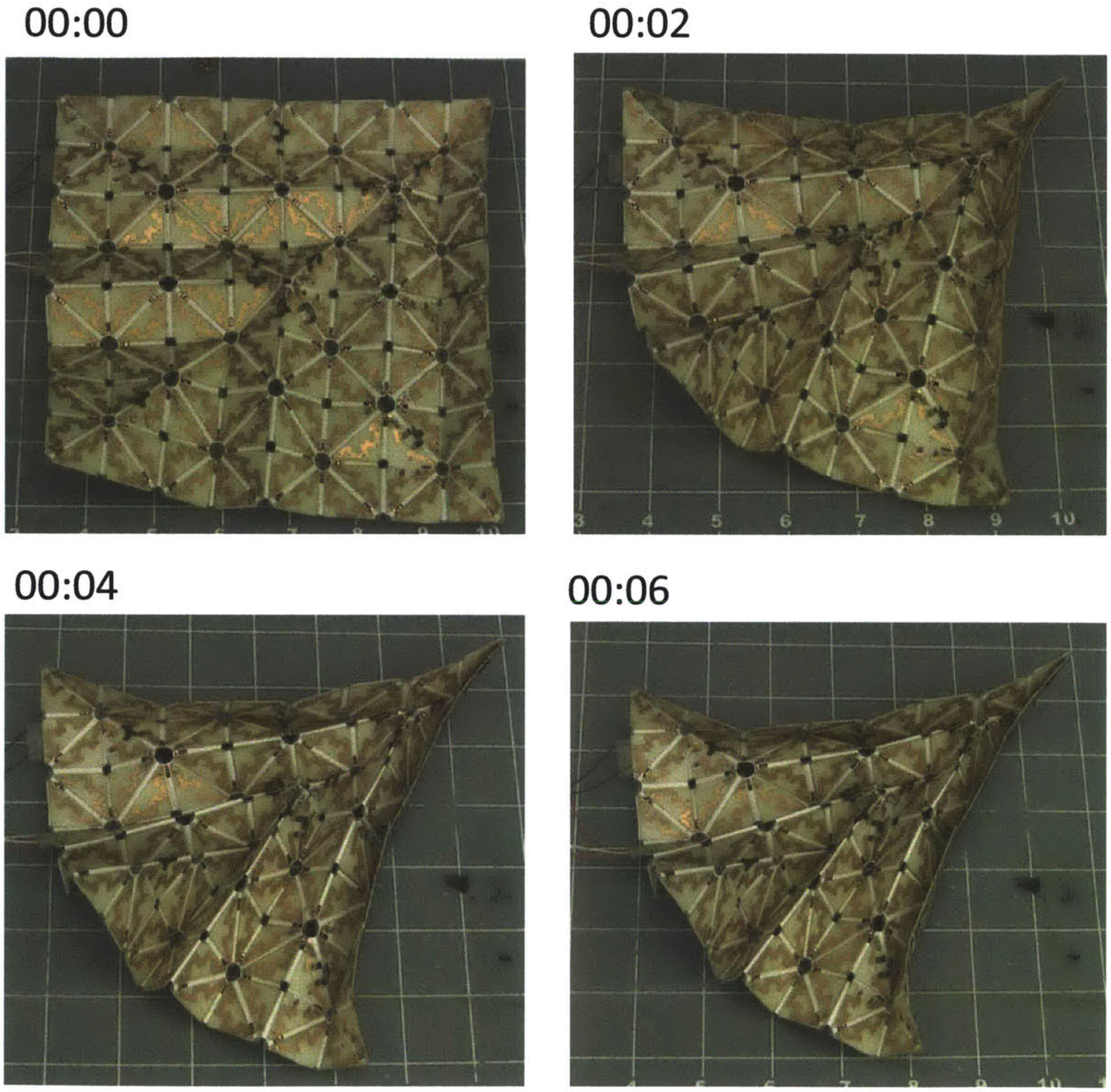
Figure 6-24: Two 8 × 8 self-folding sheet examples at the end of the self-folding operation. (a) Space Shuttle (b) Hat

6.3 Experiment with the 8×8 Self-Folding Sheet

We designed the 8 × 8 sheet to test self-folding planning for more complex shapes. We selected a space shuttle-like shape and a hat-like shape (Fig 6-24). Figures 6-25 and 6-26 show the space shuttle and hat shape transformation.

We implemented and evaluated the following four steps:

1. we generated an executable sticker design for the space shuttle and hat shapes.
2. we placed the executable sticker for the two shapes.
3. we executed the executable sticker for the space shuttle shape.
4. we executed the executable sticker for the hat shape.



00:00

00:02

00:04

00:06

Figure 6-25: Space Shuttle: 8 × 8 Self-Folding Sheet

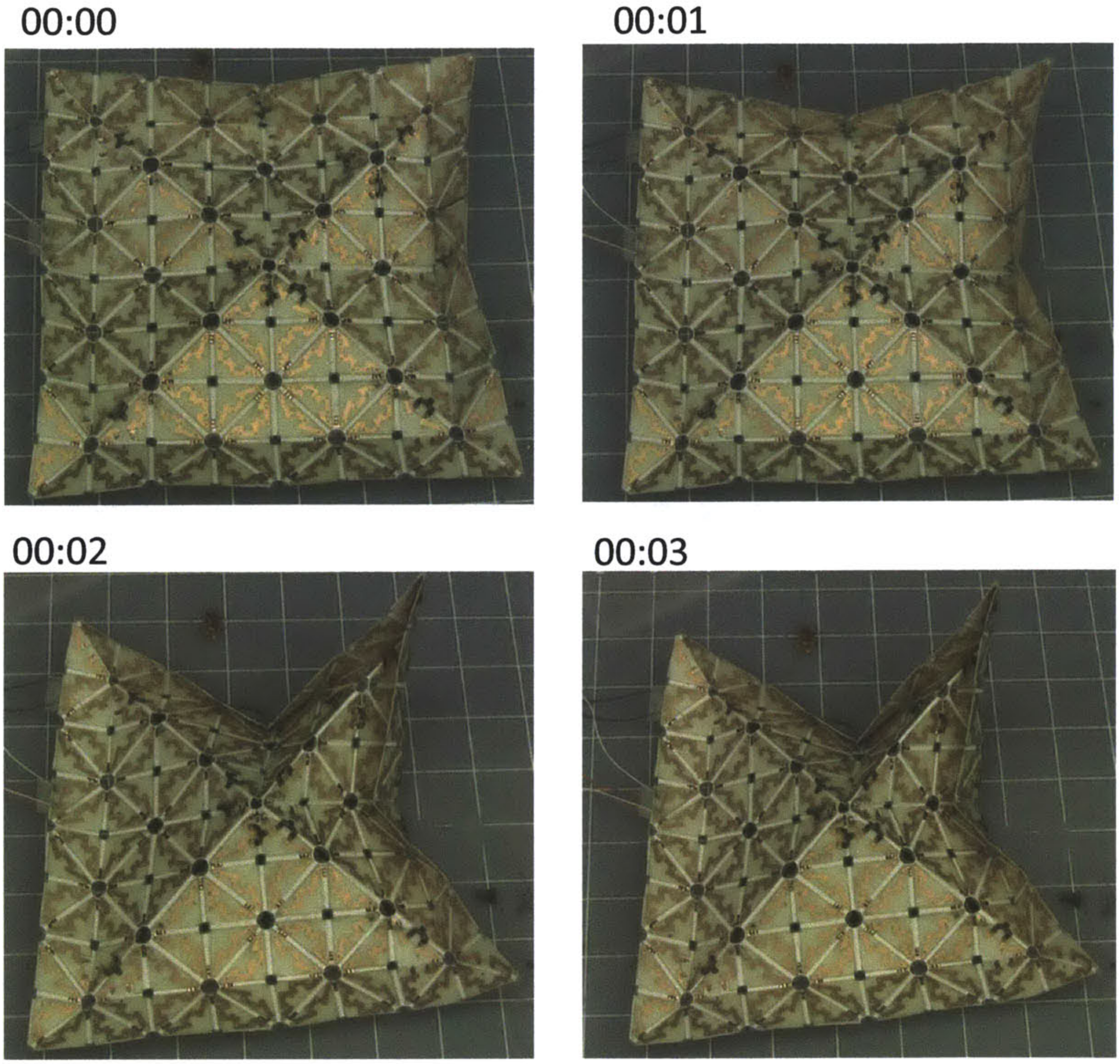


Figure 6-26: Hat: 8 × 8 Self-Folding Sheet

Table 6.9: Multiple Origami Planning Time

Single Origami Planing Analysis Time (Space Shuttle)	5.3 s (5300 ms)
Single Origami Planing Building Time (Space Shuttle)	19 ms
Single Origami Planing Analysis Time (Hat)	4.9 s (4900 ms)
Single Origami Planing Building Time (Hat)	17 ms
Multiple Origami Planning Time (with Optimization)	25 ms
Total Time	10.0 s (10261 ms)

6.3.1 Sticker Programming for Folding of Space Shuttle and Hat Shapes

We generated the executable sticker design for the two shapes with the sticker programming algorithm (Sec. 5.3). Figure 6-28 shows results of the sticker programming algorithm. The two target shapes are inputs to the algorithm (Fig. 6-29) We automatically planned the folding of the two target shapes with the origami planner and then manually computed the executable sticker design with the compiling (Fig. 5-14, Sec. 5.3.2) and linking algorithms (Fig. 5-29, Sec. 5.3.3). Figures 6-30, 6-31, and 6-32 show snapshots of origami planning for the space shuttle and hat shapes. Table 6.9 shows the planning times.

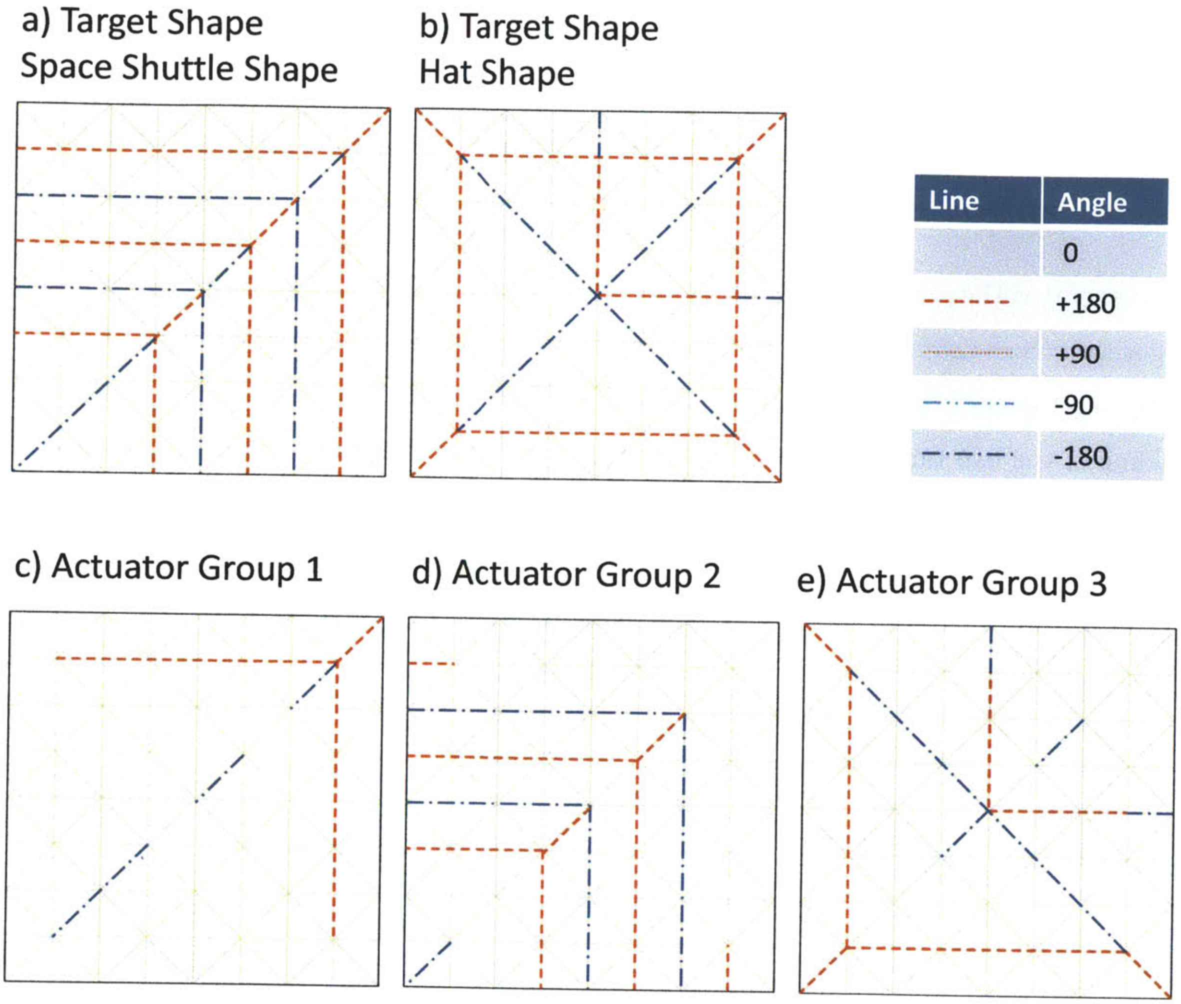


Figure 6-27: Results of origami planner (origami plan). (a)(b) are input target shapes. (c)(d)(e) are the group information of the origami plan.

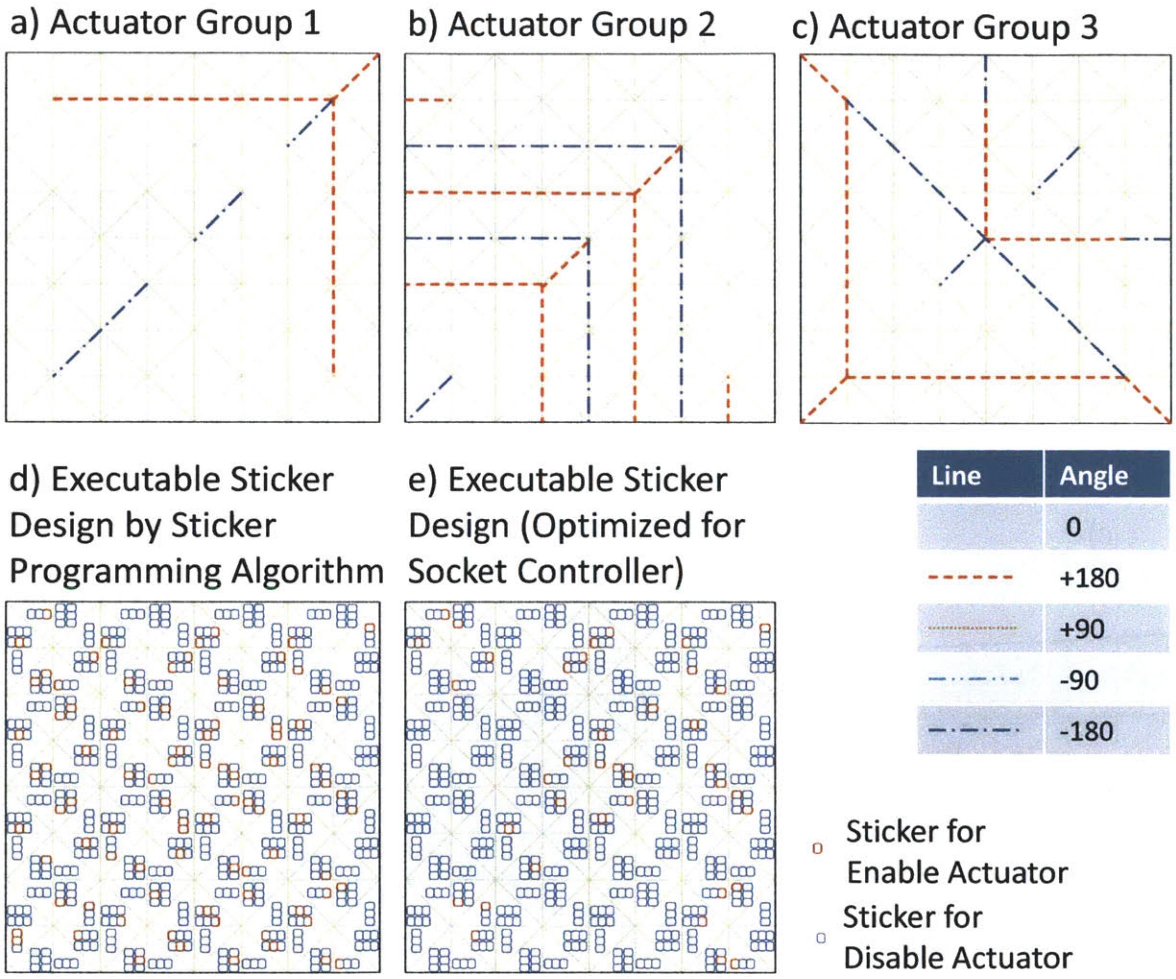

a) Actuator Group 1
b) Actuator Group 2
c) Actuator Group 3
d) Executable Sticker Design by Sticker Programming Algorithm
e) Executable Sticker Design (Optimized for Socket Controller)
Line
Angle
0
+180
+90
-90
-180
Sticker for Enable Actuator
Sticker for Disable Actuator

Figure 6-28: Results of Sticker Programming Algorithm (a)(b)(c) Origami Plan (d) Executable Sticker Design. (e) Executable Sticker Design intuitively optimized for 8 × 8 self-folding sheet. Each small square shows the sticker type for each sticker place.

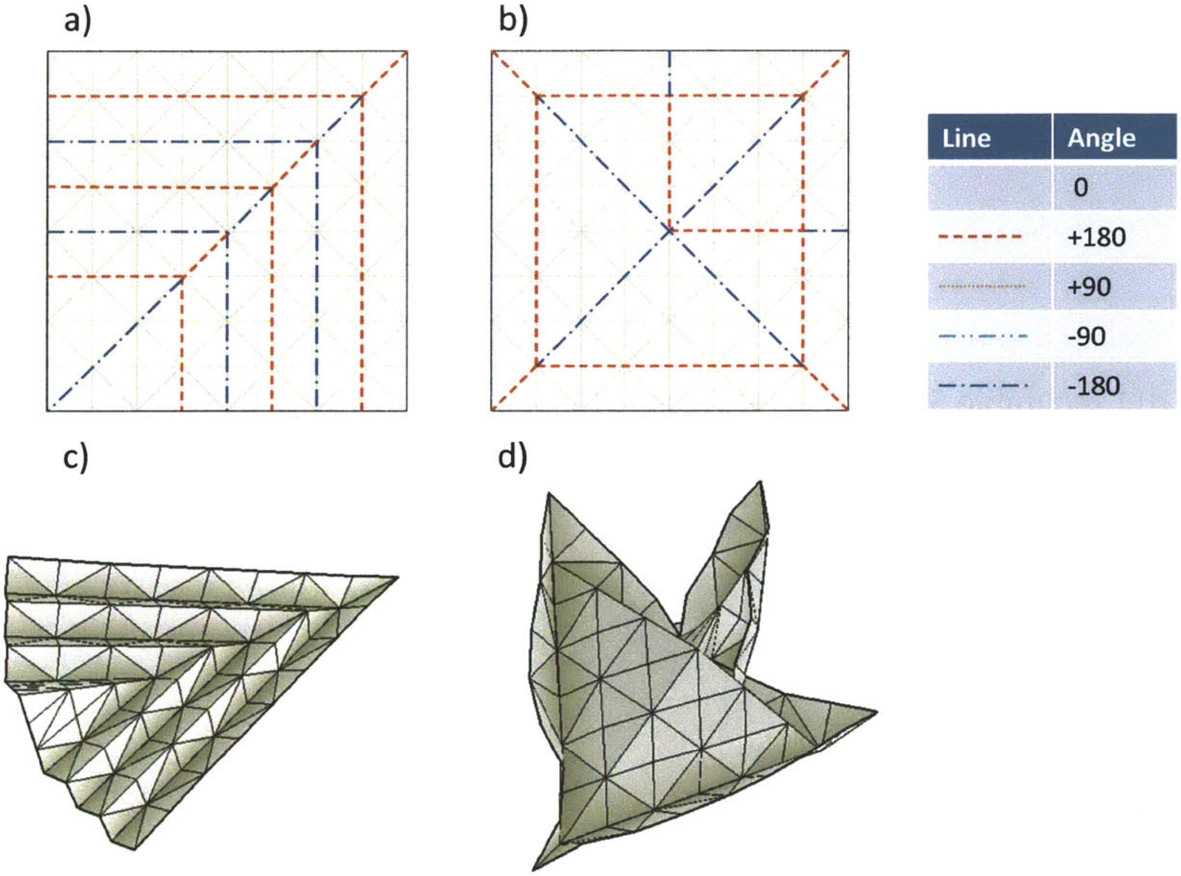


Figure 6-29: Target Shapes for Complex Motion. (a)(c) Space Shuttle (b)(d) Hat

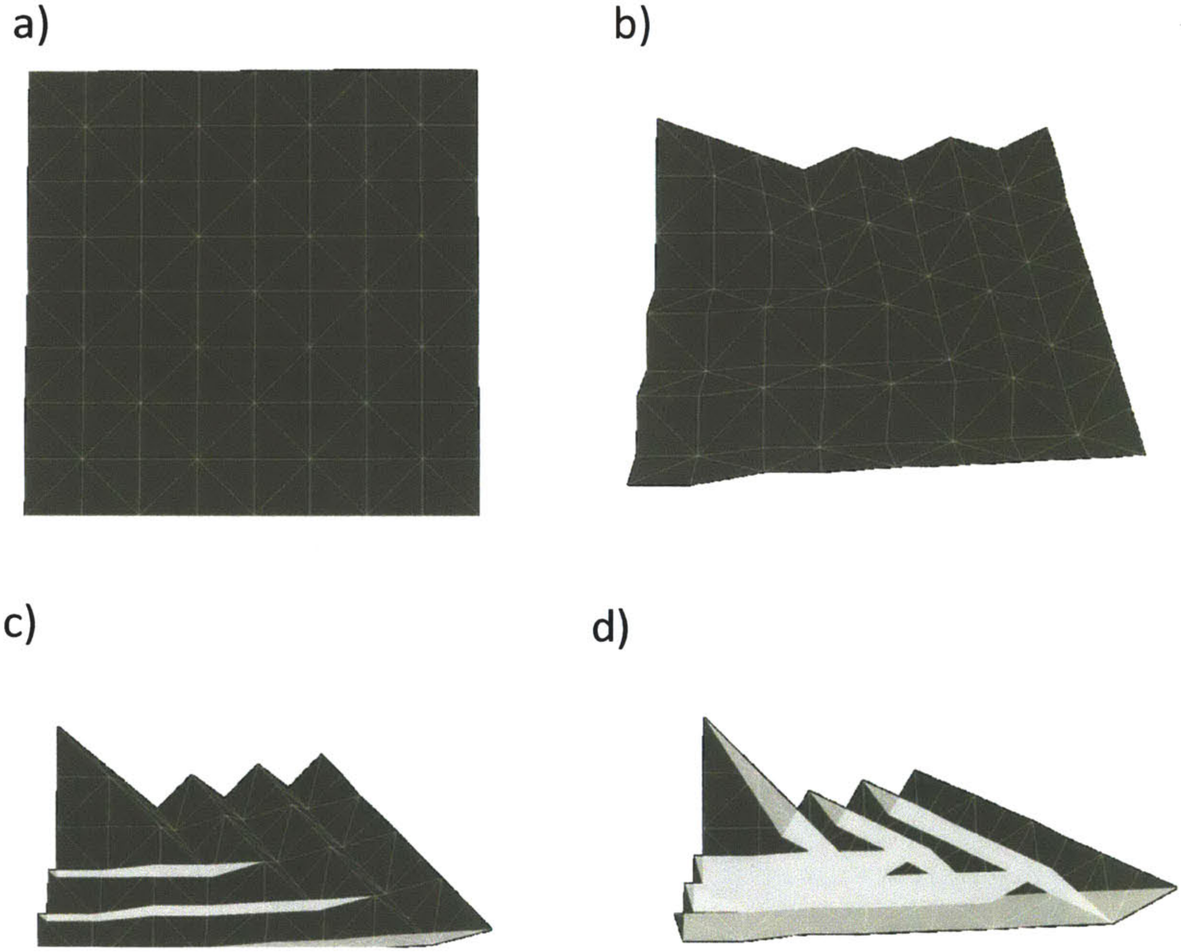


Figure 6-30: Snapshots of Space Shuttle Folding Planning [2]

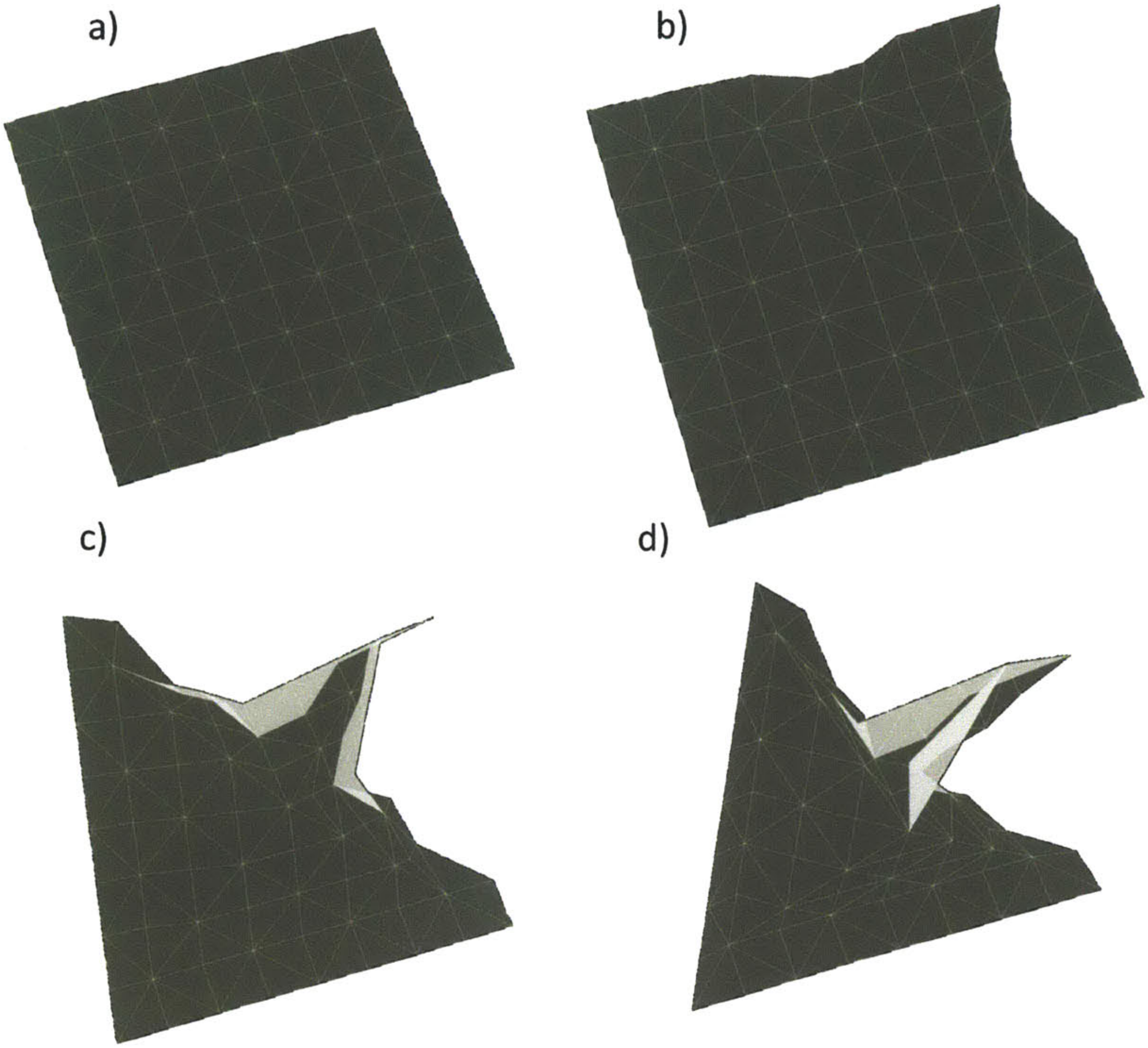


Figure 6-31: Snapshots of Hat Folding Planning. [2]

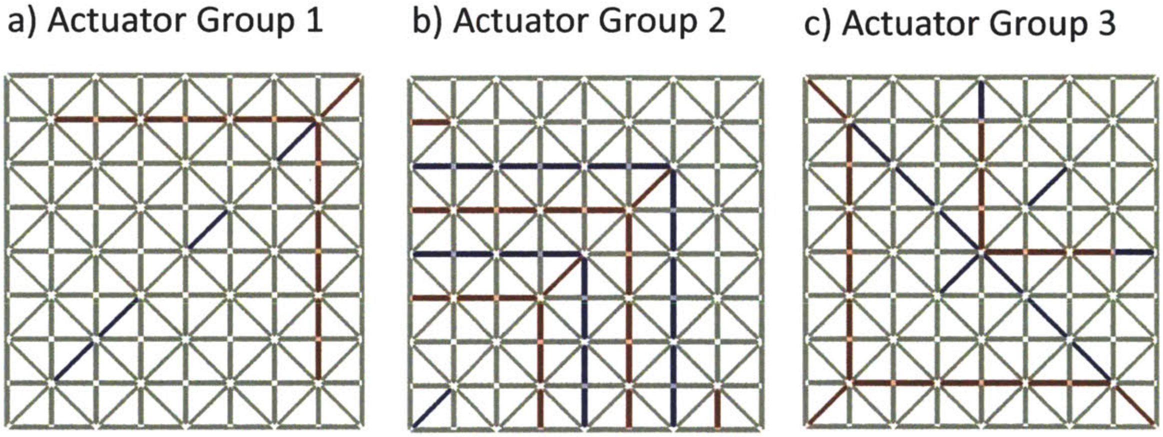


Figure 6-32: Snapshot of Sticker Program Planning. The red line denotes +180° folding. The blue line denotes -180° folding.

Table 6.10: Actuators of 8 × 8 Sheet

	Folding Actuators	Total Actuators	Total Edges	Folding Actuators / Total Actuators	Total Actuators / Total Edges
Space Shuttle (Group 1, 2)	20	36	176	55.6%	20.5%
Hat (Group 1, 3)	24	36	176	66.7%	20.5%
Total	22	36	176	61.1%	20.5%

Table 6.11: Folding Time and Current of 8 × 8 Sheet

	# of Runs	Current	Ave. Folding Time
Space Shuttle (Group1, 2)	14	5.0 A	5.9 s ±16.9%
Hat (Group1, 3)	12	5.0 A	4.5 s ±23.4%
Total	26	5.0 A	5.0 s ±19.9%

6.3.2 Results

The 8 × 8 sheet has 36 actuators and 176 edges. The socket controller controlled the 8 × 8 sheet with the relatively small number of actuators (only 20.5% of edges have the actuators). 61.1% of the actuators are used, when the sheet transformed into the both shapes (Table 6.10).

We executed the space shuttle shape folding on the 8 × 8 device 14 times. Then, we executed the folding of hat shape 12 times. The 8 × 8 sheet achieved the space shuttle and hat shapes reliably with the optimized number of actuators (Fig. 6-25, 6-26). The 8 × 8 self-folding sheet ran with current set at 5.0 A. The average folding time was 5.0 s (Table 6.11).

While we folded the 8 × 8 sheet 26 times with the two complex shapes, the experiment failed five times (Table 6.12). Like the 4 × 4 sheet, most of the failures were due to broken or weak connections between a socket and an actuator. We resolved these failures by disabling the broken actuator.

The average number of disabled actuators (for fix) was 0.81. It is 3.7 % of the folding

Table 6.12: Failure of 8 × 8 Sheet

	# of Runs	# of Failure	Ave. Failure
Space Shuttle (Group1, 2)	14	3 (of 14 runs)	2.1 (of 10 runs)
Hat (Group1, 3)	12	2 (of 12 runs)	1.6 (of 10 runs)
Total	26	5 (of 26 runs)	1.9 (of 10 runs)

Table 6.13: Disabled Actuators of 8 × 8 Sheet

	Ave. # of Disabled Actuators	Folding Actuators	Disabled Actuators / Folding Actuators	Disabled Actuators / Total Actuators
Space Shuttle (Group1, 2)	0.82	20	4.1 %	2.3 %
Hat (Group1, 3)	0.80	24	3.3 %	2.2 %
Total	0.81	22	3.7 %	2.2 %

actuators and 2.2 % of the total actuators (Table 6.13). The sheet achieved their shapes reliably with this number of the disabled actuators.

We enabled the actuator group 1 and 2 for the space shuttle-like shape. The resistance for the space shuttle shape was $17.4k\Omega$. We enabled the actuator group 1 and 3 for the hat-like shape. The resistance for the hat shape was 80.15Ω. While we executed the space shuttle shape, the average resistance of group 3 was $1.71M\Omega$. However, because we did not use the group 3 for the space shuttle shape, there was no problem to achieve the shape.

Table 6.14: Resistance of 8 × 8 Sheet

	Ave. Resistance Group1, Group2, Group3	Ave. Resistance of Folding Groups
Space Shuttle (Group1, 2)	44.8Ω, $34.8k\Omega$, $1.71M\Omega$	$17.4k\Omega$ (Group1 + Group2) / 2
Hat (Group1, 3)	109.3Ω, $14.7k\Omega$, 51.0Ω	80.15Ω (Group1 + Group3) / 2

6.4 Summary

We built 4 × 4 and 8 × 8 self-folding sheets and three sticker programs for two basic shapes and two complex shapes. We executed the programs on these self-folding sheets 53 times. The 4 × 4 sheet achieved the basic shapes reliably. The 8 × 8 sheet achieved the complex shapes reliably with the optimized number of actuators.

Chapter 7

Conclusions and Future Works

7.1 Conclusions

We have described a programming method including a hardware design and a suite of algorithms for controlling micro-thin sheets with built-in creases and embedded actuators, and connectors. We described the hardware design for controlling the self-folding sheet (smart sheet) for the automatic transformation of self-folding sheets into multiple objects. We described the details of the algorithms that automatically create programs for the automatic transformation of multiple target shapes from a single sheet. The algorithms are designed for the sheet containing the implementation of the hardware design. Finally we developed two different hardware devices and conducted experiments with the sticker placement and self-folding control algorithms. We achieved four target shapes reliably. We collected and analyzed self-folding data during these experiments.

7.2 Future Works

In the future, we need to consider how to enhance the design of the sticker controller in order to have it deliver more complex computation. The sticker controller we described is a state machine that computes its final status (final shape) using laws of physics and quantities such as resistant force, gravity, or torque. We would like to explore how the sticker controller could be viewed as a machine.

The algorithms for the sticker programming are centralized and computed off-board. While our previous method for self-folding control [14] did not have a programming ability, the approach in this thesis brings a rudimentary programming capability to the self folding sheet. For the next step, we will examine the possibility of on-board programming algorithms.

Appendix A

Video: Sticker Controller and Sticker Programming

http://www.drancom.com/smthesis/video/an_st_prog.mov

Appendix B

Design of Self-Folding Sheet

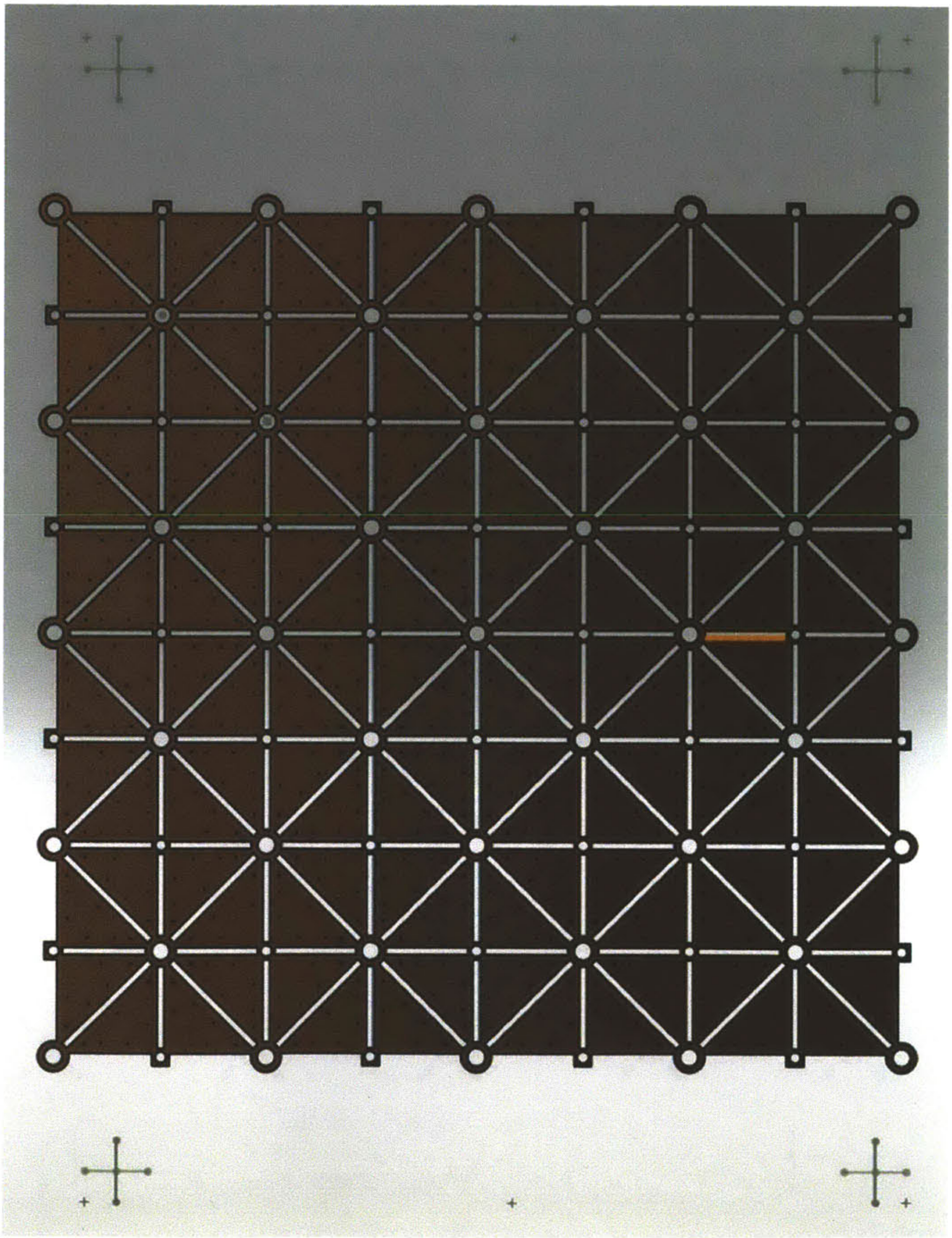
Common Parts

The Body
The Actuator

The Body

(192mm x 192mm)

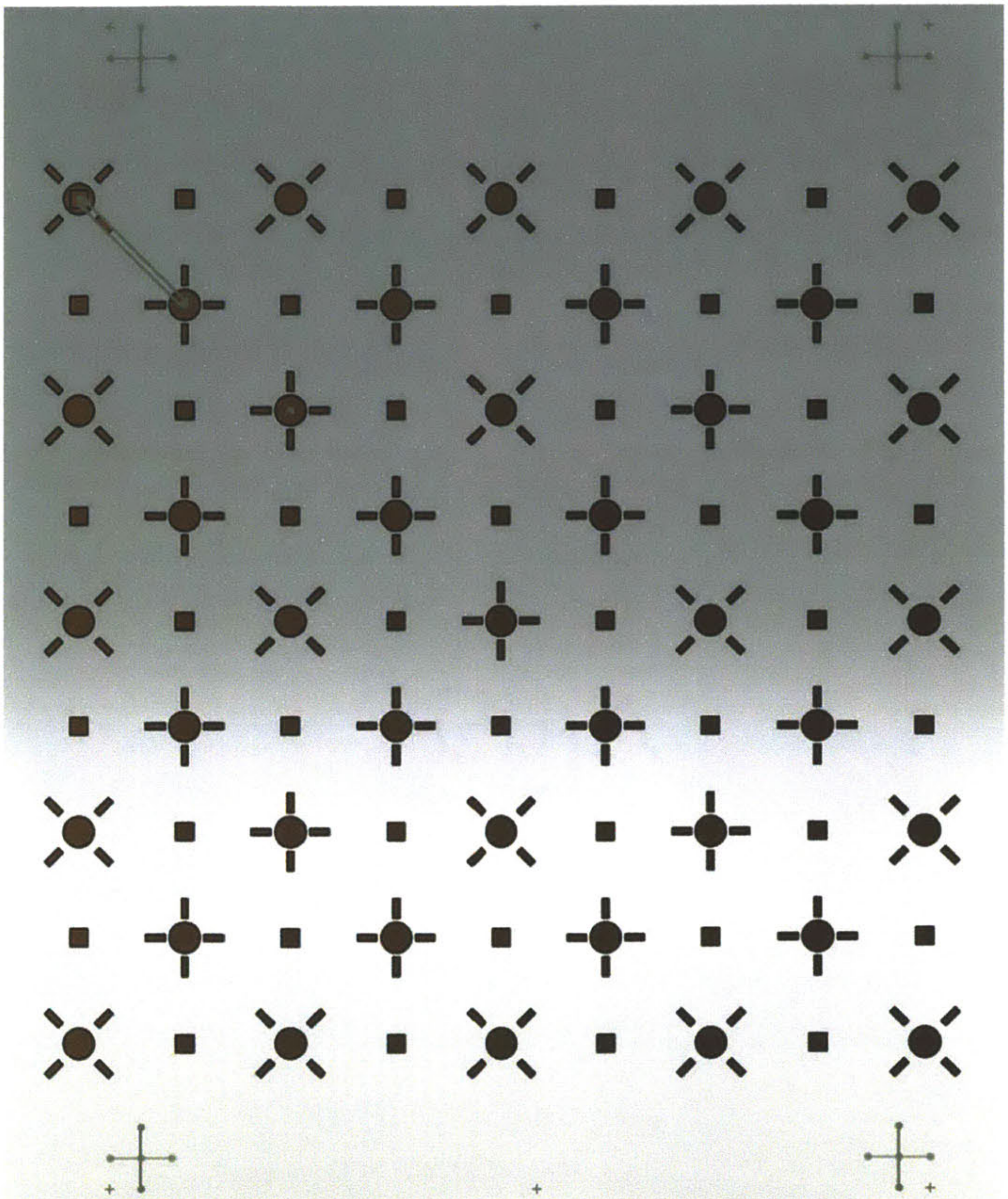
Tiles



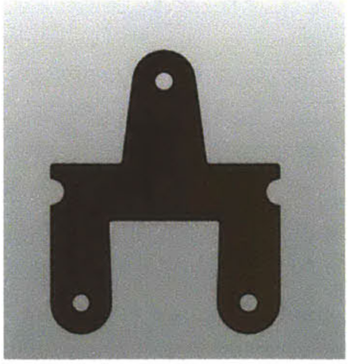
The Body

(192mm x 192mm)

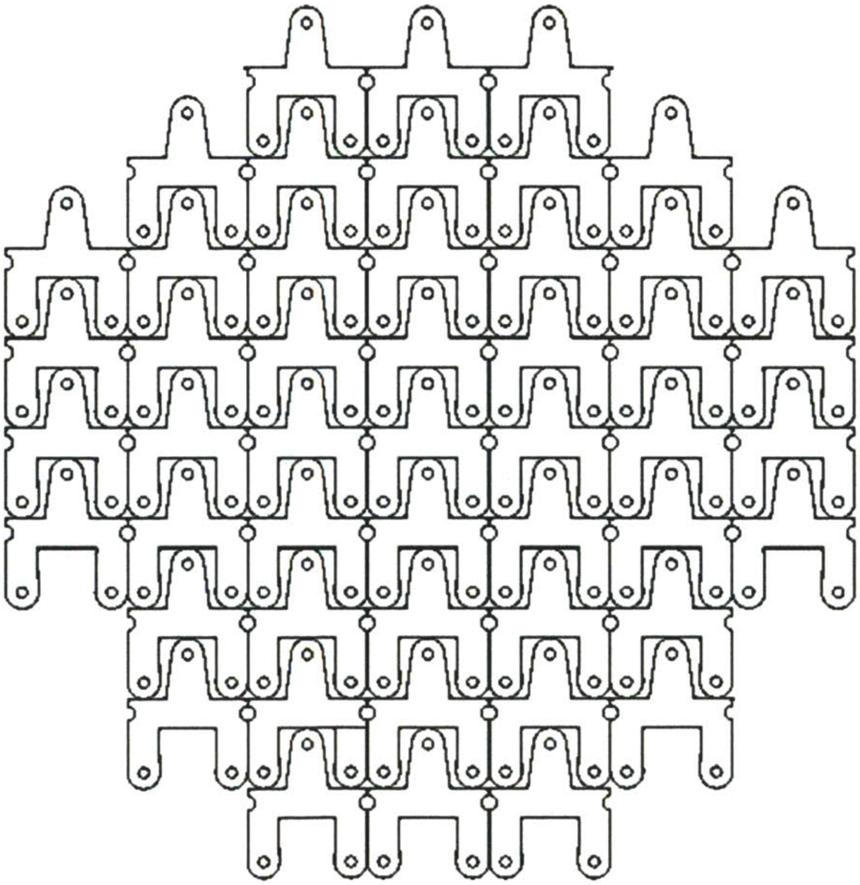
Joints



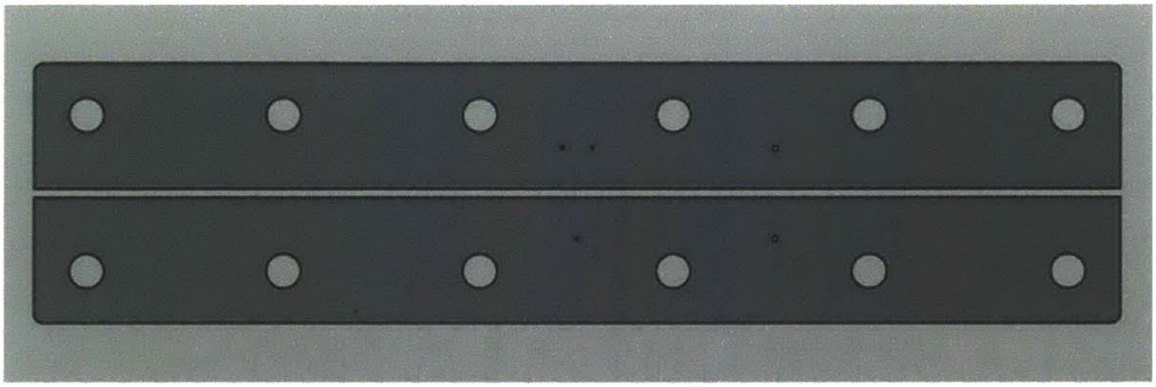
The Actuator

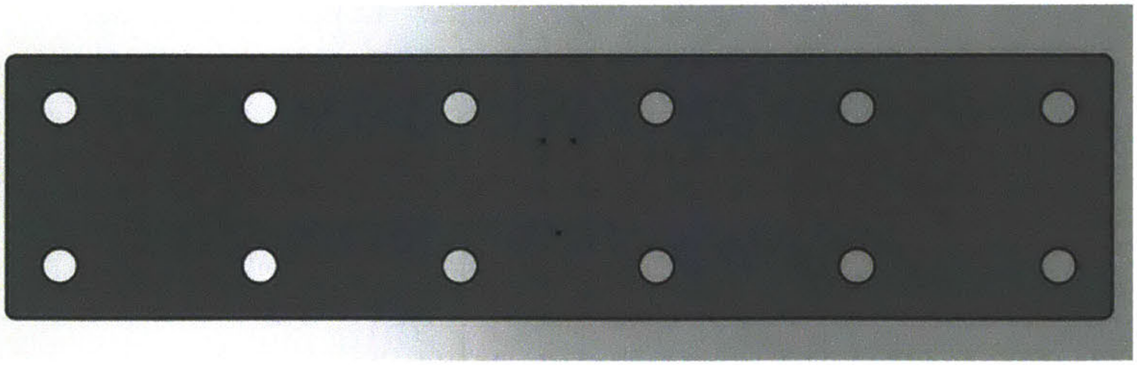


(10.23mmx8.21mm)



The Gig





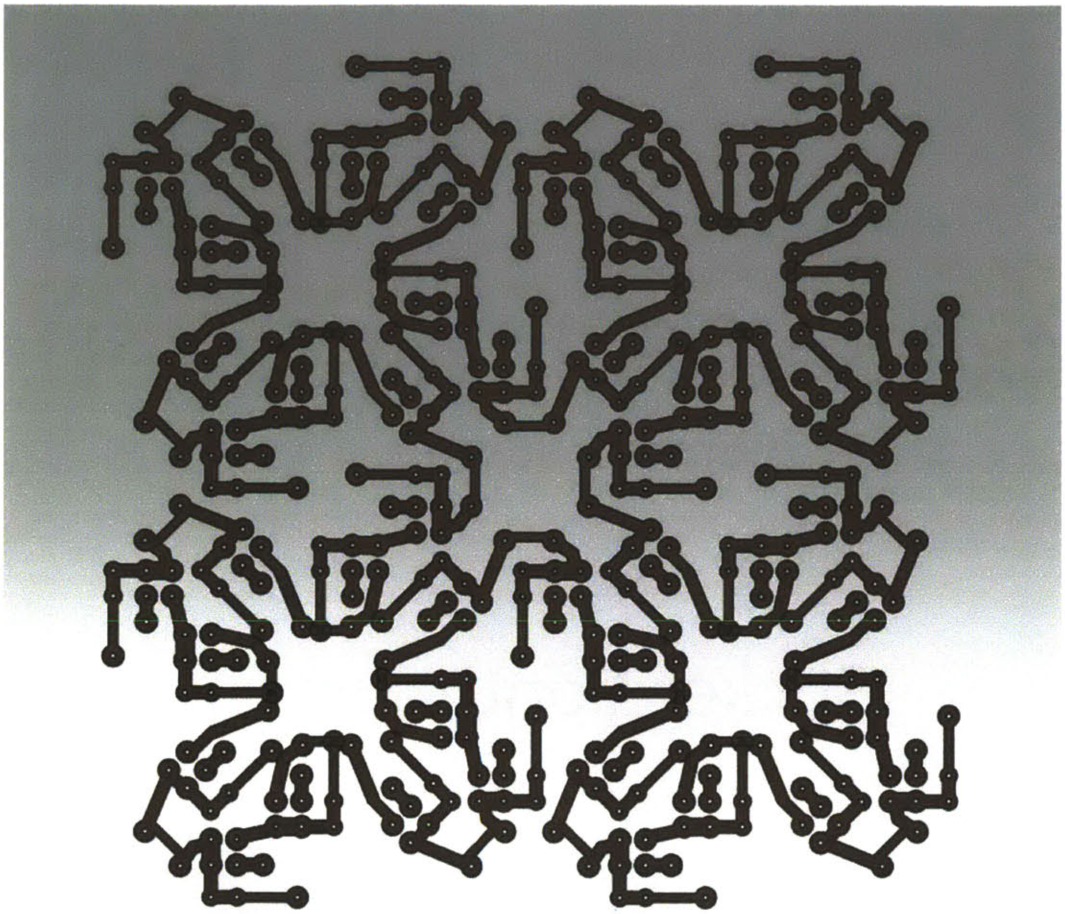
4x4 Self-Folding Sheet

(96mm x 96mm)

Sticker Controller

The Circuit

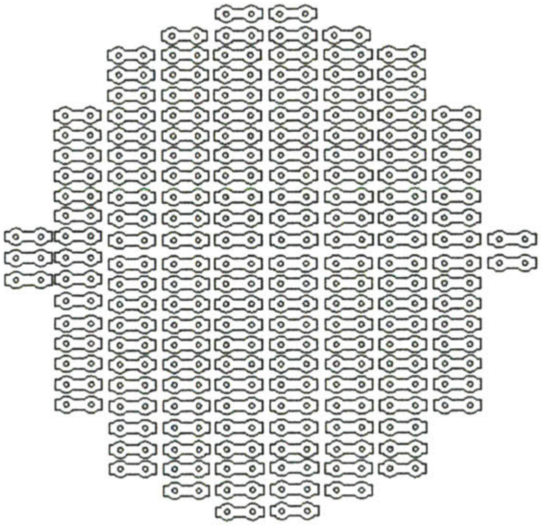
Main



Socket for Bottom

The Circuit

Patches for Executable Sticker



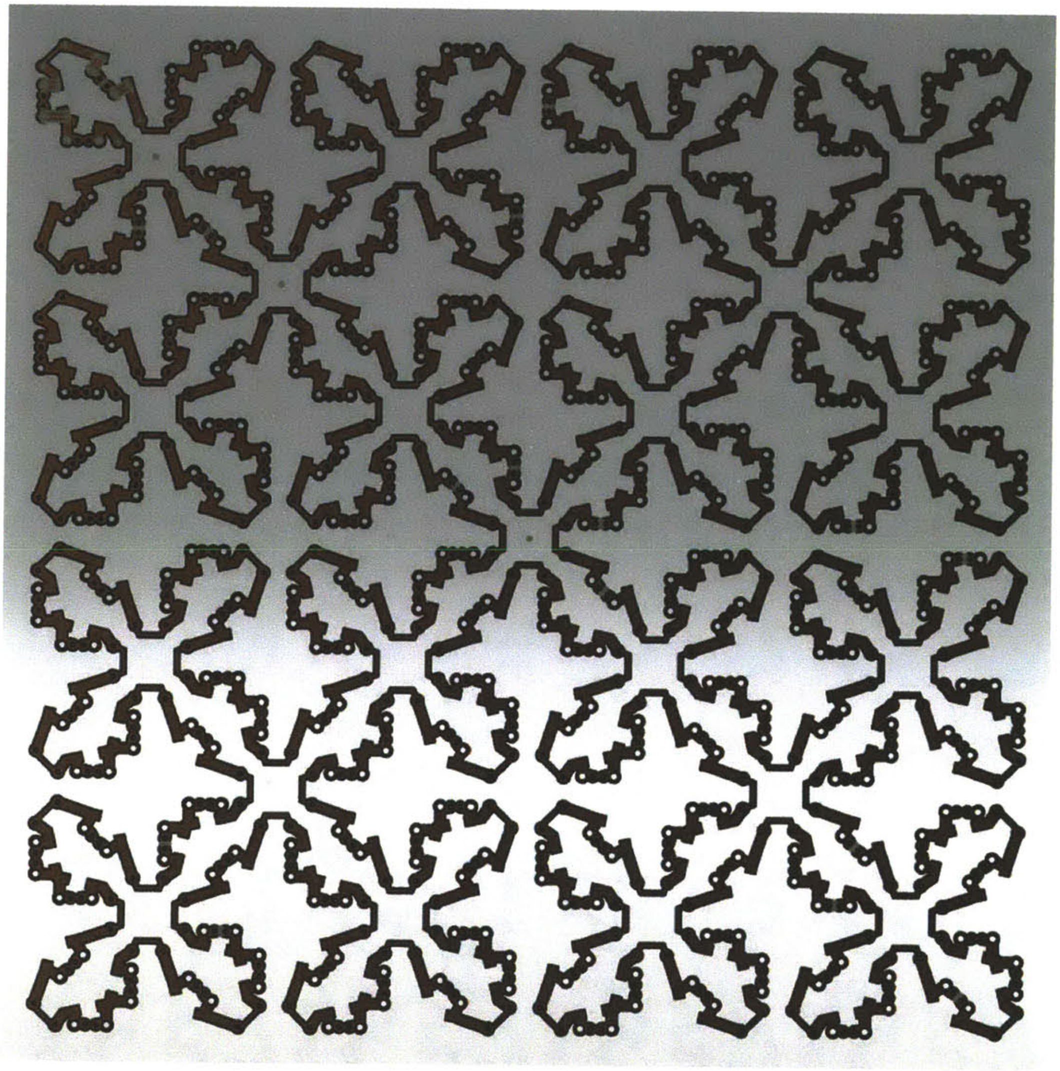
8x8 Self-Folding Sheet

(192mm x 192mm)

Sticker Controller

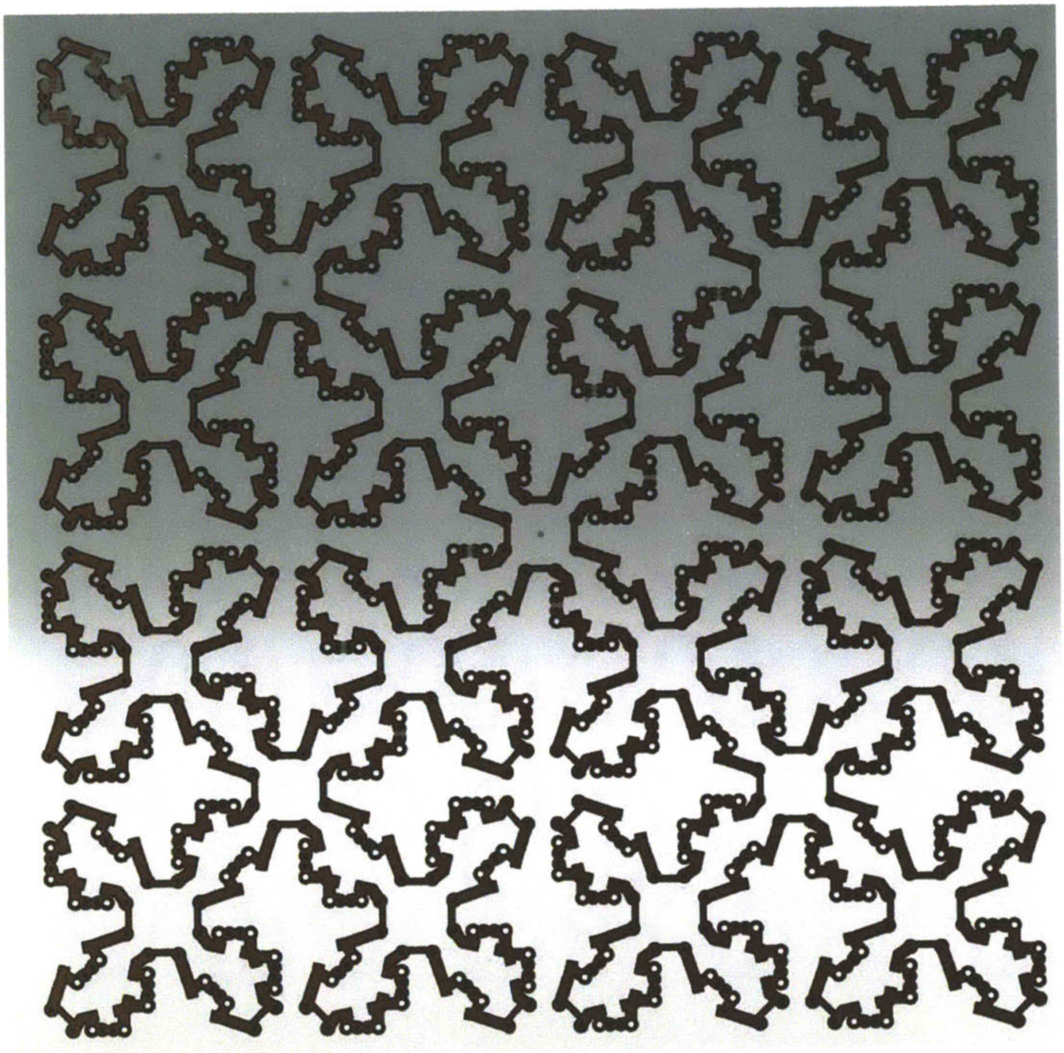
The Circuit

Layer 1



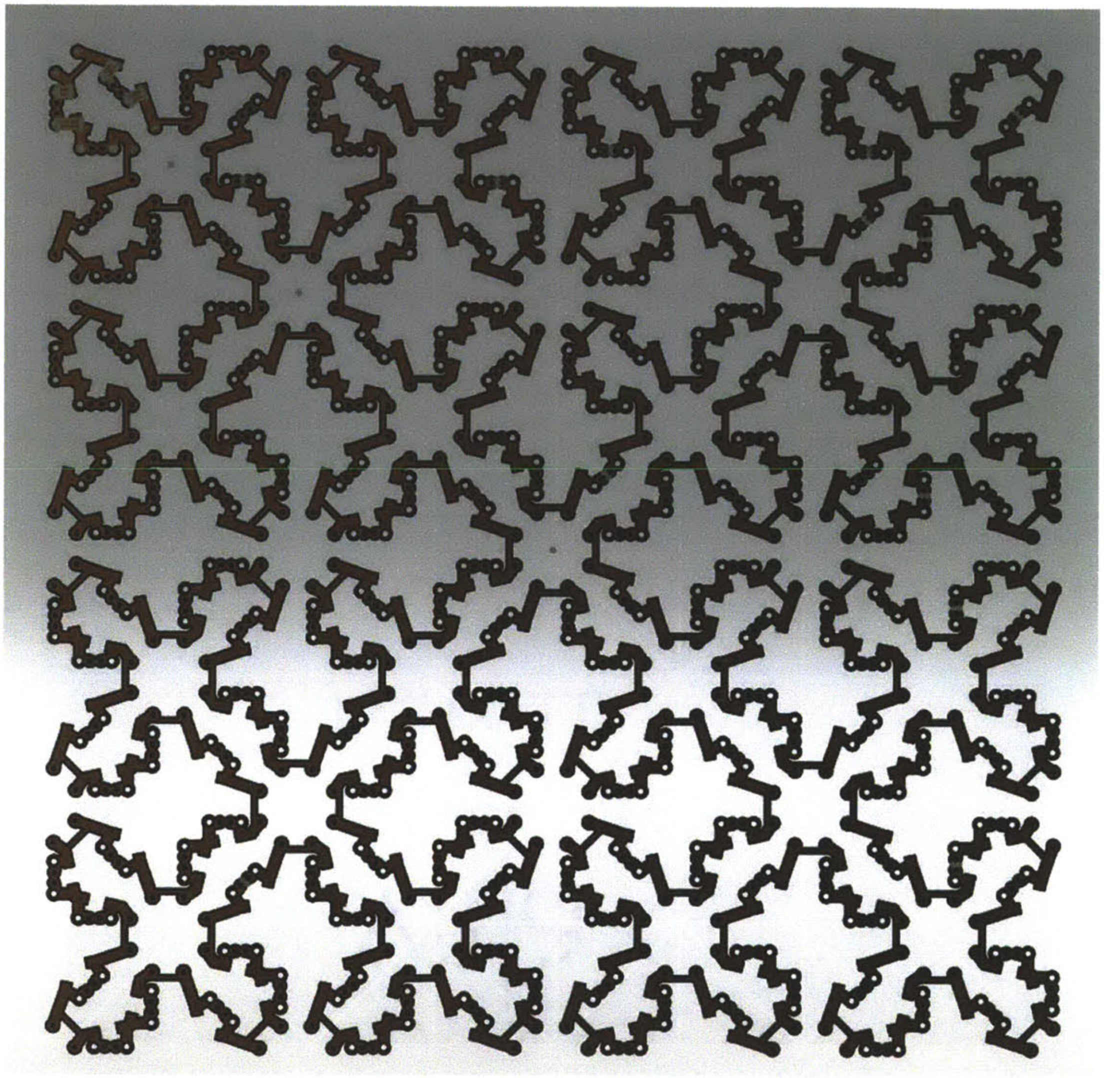
The Circuit

Layer 2

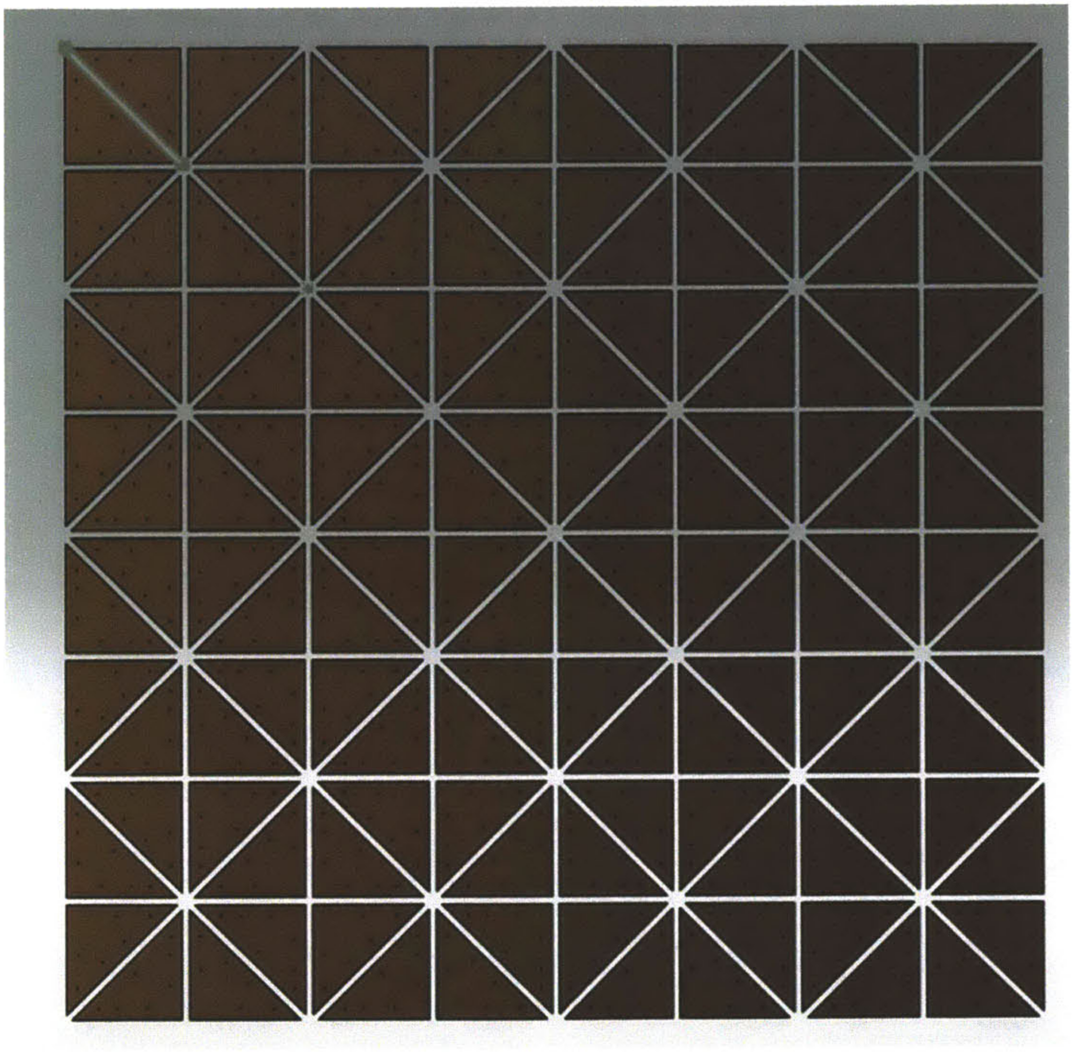


The Circuit

Layer 3



The Cover and The Insulator



Appendix C

Source Codes

```java
package com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigami;

import sun.security.util.PendingException;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FilePlan;
import com.drancom.programmableMatter.folding.dataFile.FilePlanForWiring;
import com.drancom.programmableMatter.folding.origami.planner.Plan;
import com.drancom.programmableMatter.folding.origami.planner.Planner;

public class PlanerForOrigami implements Planner {
//   Default: percentage of level0 = 33, percentage of level1 = 66,
      percentage of level2 = 100
//   NoiseNumber = 0.01

  final static float PERCENTAGE_OF_LEVEL0 = 0.33f;
  final static float PERCENTAGE_OF_LEVEL1 = 0.66f;
  final static float PERCENTAGE_OF_LEVEL2 = 1.0f;

  final static float NOISE_NUMBER= 0.01f;

  PlanForOrigami planForOrigami;

  @Override
  public void build(Paper[] papers) {

//     Input:  Angledata[][]
    float angleData[][] = new float[papers[0].getNumberOfEdges()][papers.
         length];
//     Output: Plan[phase][numberOfEdge], numberOfPhase
    this.planForOrigami = new PlanForOrigami();
    int plan[][] = new int[papers.length][papers[0].getNumberOfEdges()];
    int numberOfPhases;

    float max[] = new float [papers[0].getNumberOfEdges()];
    float min[] = new float [papers[0].getNumberOfEdges()];
    float standard[] = new float [papers[0].getNumberOfEdges()];


    int angleLevel[][] = new int [papers[0].getNumberOfEdges()][papers.
         length];
    int angleDifference[][] = new int [papers[0].getNumberOfEdges()][papers
         .length];

    int phase;
    boolean isBuildingPlan;
    boolean isAllZero;

    int i;
    int j;
    int k=0;

    for (i=0; i < papers[0].getNumberOfEdges(); i++){
      for (j=0; j < papers.length; j++) {
        angleData[i][j] = papers[j].getLine(i).getAngle();
        angleLevel[i][j] = 0;
        if (angleData[i][j] <0) {
          k++;
        }
      }
    }

//     1.  If ABS(angledata[0..n][0..t]) < NoiseNumber), angledata[0..n][0..
      t] <- 0
    for (i=0; i < papers[0].getNumberOfEdges(); i++){
      for (j=0; j < papers.length; j++) {
        if(Math.abs(angleData[i][j]) < NOISE_NUMBER) {
          angleData[i][j] = 0.0f;
        }
      }
    }


//     2.  Get a Max[0..n] from edges angles.
//     3.  Get a Min[0..n] from edges angles.
    for (i=0; i< papers[0].getNumberOfEdges(); i++) {
      max[i] = 0.0f;
      min[i] = 0.0f;
      standard[i] = 0.0f;
    }

    for (i=0; i < papers[0].getNumberOfEdges(); i++){
      for (j=0; j < papers.length; j++) {
        if(max[i] < angleData[i][j]) {
          max[i] = angleData[i][j];
        }
        if(min[i] > angleData[i][j]) {
          min[i] = angleData[i][j];
        }
      }
    }


//     4.  If ABS(Max[0..n]) > ABS(MIN[0..n]), Standard[0..n] <- Max[0..n];
      otherwise, Standard[0..n] <- Min[0..n]
    for (i=0; i < papers[0].getNumberOfEdges(); i++){
      if(Math.abs(max[i]) > Math.abs(min[i])) {
        standard[i] = max[i];
      } else {
        standard[i] = min[i];
      }
    }

//     5.  Make Anglelevel[n][t] from AngleData[n][t]:
//         If AngleData[i][j] = 0 or Standard[i] = 0, AngleLevel[i][j] = 0;
//         otherwise, If Standard[i] > 0, AngleLevel[I][j] = 2
//                If AngleData[i][j] < Standard[i] * percentage of level1,
      AngleLevel[I][j] = 1
//                    If AngleData[i][j] < Standard[i] * percentage of
      level0, AngleLevel[I][j] = 0
//         otherwise, If Standard[i] < 0, AngleLevel[I][j] = -2
//                If AngleData[i][j] > Standard[i] * percentage of level1,
      AngleLevel[I][j] = -1
//                    If AngleData[i][j] > Standard[i] * percentage of
      level0, AngleLevel[I][j] = 0

    for (i=0; i < papers[0].getNumberOfEdges(); i++){
      for (j=0; j < papers.length; j++) {
        if (angleData[i][j] == 0.0f || standard[i] == 0.0f) {
          angleLevel[i][j] = 0;
        } else if (standard[i] > 0 ) {
          angleLevel[i][j] = 2;
          if (angleData[i][j] < standard[i] * PERCENTAGE_OF_LEVEL1) {
            angleLevel[i][j] = 1;
            if(angleData[i][j] < standard[i] * PERCENTAGE_OF_LEVEL0) {
              angleLevel[i][j] = 0;
            }
          }
        } else if (standard[i] < 0 ) {
          angleLevel[i][j] = -2;
          if (angleData[i][j] > standard[i] * PERCENTAGE_OF_LEVEL1) {
            angleLevel[i][j] = -1;
            if(angleData[i][j] > standard[i] * PERCENTAGE_OF_LEVEL0) {
              angleLevel[i][j] = 0;
            }
          }
        }
      }
    }

//      6.  If AngleLevel[i][j] = 1, AngleDifference[i][j] = 1;
//          If AngleLevel[i][j] > 1 and AngleLevel[i][j] - AngleLevel[i][j+1]
        != 0, AngleDifference[i][j] = 1
//          Otherwise, AngleDifference[i][j] = 0;
//      If AngleLevel[i][j] = -1, AngleDifference[i][j] = -1;
//      If AngleLevel[i][j] < 1 and AngleLevel[i][j] - AngleLevel[i][j+1] !=
        0, AngleDifference[i][j] = -1
//      Otherwise, AngleDifference[i][j] = 0

    for (i=0; i < papers[0].getNumberOfEdges(); i++){
      for (j=0; j < papers.length-1; j++) {

        angleDifference[i][j] = 0;
        if (angleLevel[i][j] == 1) {
          angleDifference[i][j] = 1;
        }
        if (angleLevel[i][j] > 0 && angleLevel[i][j] - angleLevel[i][j+1]
            != 0 ) {
          angleDifference[i][j] = 1;
        }
        if (angleLevel[i][j] == -1) {
          angleDifference[i][j] = -1;
        }
        if (angleLevel[i][j] < 0 && angleLevel[i][j] - angleLevel[i][j+1]
            != 0 ) {
          angleDifference[i][j] = -1;
        }
      }
    }

//      7.  Phase <- 0
//      8.  numberOfPhase
//      9.  isBuildingPlan <- false
//      10. IsAllZero <- true
    phase = 0;
    numberOfPhases = 1;
    isBuildingPlan = false;
    isAllZero = true;

//      11. Plan[1..][1..numberOfEdge] <- 0
    for (i=0; i < papers.length; i++) {
      for (j=0; j < papers[0].getNumberOfEdges(); j++){
        plan[i][j] = 0;
      }
    }

//      12. For i = lastTime to 1
    for (i=papers.length -1 ; i>=0; i--) {


//      13.         IsAllZero = true
      isAllZero = true;
//      14.         For j = 1 to numberOfEdge
      for (j=0; j< papers[0].getNumberOfEdges(); j++) {
//      15.                 If AngleDifference[j][i] = -1,
        if (angleDifference[j][i] == -1){
//      16.                         Do Plan[Phase][j] = -1
//      17.                         isBuildingPlan = true
//      18.                         IsAllZero = false
          plan[phase][j] = -1;
          isBuildingPlan = true;
          isAllZero = false;
        }
//      19.                 If AngleDifference[j][i] = 1,
        if (angleDifference[j][i] == 1){
//      20.                         Do Plan[Phase][j] = 1
//      21.                         isBuildingPlan = true
//      22.                         IsAllZero = false
          plan[phase][j] =  1;
          isBuildingPlan = true;
          isAllZero = false;
        }
      }
//      23.         If isBuildingPlan = true and isAllzero = true
      if (isBuildingPlan == true && isAllZero == true) {
//      24.                 Do lastPhase = Phase
//      25.                 Phase++
//      26.                 isBuildingPlan = false
        phase++;
        numberOfPhases  = phase;
        isBuildingPlan = false;
      }
    }

    int temp_plan[][] = new int[numberOfPhases + 3][papers[0].
        getNumberOfEdges()];
    for (i = 0; i < numberOfPhases; i++) {
      for(j=0; j < papers[0].getNumberOfEdges() ; j++){
        temp_plan[i][j] = plan[i][j];
      }
    }

    this.planForOrigami.setPlanTable(temp_plan);
    this.planForOrigami.setNumberOfEdges(papers[0].getNumberOfEdges());
    this.planForOrigami.setNumberOfPhases(numberOfPhases);
  }

  @Override
  public void exportPlan(String fileName, Paper[] papers) {
    FilePlan filePlan = new FilePlanForOrigami();

    filePlan.build(fileName, planForOrigami, papers);
  }

  @Override
  public Plan getPlan() {
    // TODO Auto-generated method stub
    return this.planForOrigami;
  }

  @Override
  public void build(Plan[] plans) {
    // TODO Auto-generated method stub

  }

  @Override
  public void exportPlan(String fileName) {
    // TODO Auto-generated method stub

  }
}
```

```
package com.drancom.programmableMatter.folding.simulator.
    simulatorForOrigami;

import com.drancom.programmableMatter.folding.controller.paper.Line;
import com.drancom.programmableMatter.folding.dataFile.FilePlan;
import com.drancom.programmableMatter.folding.origami.planner.Plan;

public class PlanForOrigami implements Plan {
  float edgeTable[][]; // [edge number] [ x0 y0 x1 y1]
  int planTable[][];   // [phase] [ edge number]


  int numberOfEdges;
  int numberOfPhases;

  int angleOfRotation = 0;

  boolean isInvert = false;

  public int[][] getPlanTable(){
    return planTable;
  }

  public float[][] getEdgeTable() {
    return edgeTable;
  }

  public void setPlanTable(int[][] planTable){
    int i;
    int j;

    this.planTable = new int [planTable.length][planTable[0].length];

    for (i=0; i<planTable.length ; i++) {
      for(j=0; j< planTable[i].length ; j++) {
        this.planTable[i][j] = planTable[i][j];
      }
    }
  }

  public void setEdgeTable(float[][] edgeTable){
    int i;
    int j;

    this.edgeTable = new float[edgeTable.length][edgeTable[0].length];

    for (i=0; i < edgeTable.length ; i++) {
      for(j=0; j < edgeTable[i].length ; j++) {
        this.edgeTable[i][j] = edgeTable[i][j];
      }
    }
  }

  public int getNumberOfPhases() {
    return numberOfPhases;
  }

  public void setNumberOfPhases(int numberOfPhases) {
    this.numberOfPhases = numberOfPhases;
  }

  public int getNumberOfEdges() {
    return numberOfEdges;
  }

  public void setNumberOfEdges(int numberOfEdges) {
    this.numberOfEdges = numberOfEdges;
  }


  public void invert() {
    int i;
    int j;

    for (i=0; i<planTable.length ; i++) {
      for(j=0; j< planTable[i].length ; j++) {
        this.planTable[i][j] = planTable[i][j] * -1;
      }
    }
    if (isInvert) {
      isInvert = false;
```

```
    } else {
      isInvert = true;
    }
  }

  public boolean isInvert(){
    return isInvert;
  }

  public void rotateClockwise() {
    int i;

    for (i=0; i<edgeTable.length ; i++) {
      this.edgeTable[i][0] = -1 * edgeTable[i][1]; // x0' = -y0
      this.edgeTable[i][1] =      edgeTable[i][0] + 1; // y0' =  x0 + 1
      this.edgeTable[i][2] = -1 * edgeTable[i][3]; // x1' = -y1
      this.edgeTable[i][3] =      edgeTable[i][2] + 1; // y1' =  x1 + 1
    }
    angleOfRotation += 90;
    angleOfRotation %= 360;

    sort();
  }

  public void rotateCounterclockwise() {
    int i;

    for (i=0; i<edgeTable.length ; i++) {
      this.edgeTable[i][0] =  1 * edgeTable[i][1] + 1 ; // x0' = -y0
      this.edgeTable[i][1] = -1 * edgeTable[i][0]; // y0' =  x0 + 1
      this.edgeTable[i][2] =  1 * edgeTable[i][3] + 1; // x1' = -y1
      this.edgeTable[i][3] = -1 * edgeTable[i][2]; // y1' =  x1 + 1
    }
    angleOfRotation += 270;
    angleOfRotation %= 360;

    sort();
  }

  public void sort() {

    int i;
    int j;
    int k;

    float x0,y0;
    float x1,y1;
    float edgeData;
    int   tempPlanData;

    // sort startPoint and endPoint

    for (i=0; i < numberOfEdges; i++){
      x0 = edgeTable[i][0];
      y0 = edgeTable[i][1];
      x1 = edgeTable[i][2];
      y1 = edgeTable[i][3];

      if (x0 == x1) {
        if (y0 < y1) {

          edgeTable[i][0] = x0;
          edgeTable[i][1] = y0;
          edgeTable[i][2] = x1;
          edgeTable[i][3] = y1;

        } else {

          edgeTable[i][0] = x1;
          edgeTable[i][1] = y1;
          edgeTable[i][2] = x0;
          edgeTable[i][3] = y0;

        }
      } else if (y0 == y1) {
        if (x0 < x1) {

          edgeTable[i][0] = x0;
          edgeTable[i][1] = y0;
          edgeTable[i][2] = x1;
          edgeTable[i][3] = y1;

        } else {
          edgeTable[i][0] = x1;
          edgeTable[i][1] = y1;
          edgeTable[i][2] = x0;
          edgeTable[i][3] = y0;

        }

      } else if (x0 < x1) {
        edgeTable[i][0] = x0;
        edgeTable[i][1] = y0;
        edgeTable[i][2] = x1;
        edgeTable[i][3] = y1;

      } else {

        edgeTable[i][0] = x1;
        edgeTable[i][1] = y1;
        edgeTable[i][2] = x0;
        edgeTable[i][3] = y0;

      }

    }

    // sort lines
    for (i = 0; i < numberOfEdges - 1; i++) {
      for (j = i + 1; j < numberOfEdges; j++) {

        if (edgeTable[i][0] >
          edgeTable[j][0] ) {

          // swap edge
          for (k = 0; k < 4 ; k++) {
            edgeData = edgeTable[i][k];
            edgeTable[i][k] = edgeTable[j][k];
            edgeTable[j][k] = edgeData;
          }

          // swap planTable
          for (k = 0 ; k < numberOfPhases; k++) {
            tempPlanData =  planTable[k][i];
            planTable[k][i] = planTable[k][j];
            planTable[k][i] = tempPlanData;
          }

        } else if (edgeTable[i][0] ==
              edgeTable[j][0] ) {
          if (edgeTable[i][1] >
            edgeTable[j][1] ) {
            // swap edge
            for (k = 0; k < 4 ; k++) {
              edgeData = edgeTable[i][k];
              edgeTable[i][k] = edgeTable[j][k];
              edgeTable[j][k] = edgeData;
            }

            // swap planTable
            for (k = 0 ; k < numberOfPhases; k++) {
              tempPlanData =  planTable[k][i];
              planTable[k][i] = planTable[k][j];
              planTable[k][i] = tempPlanData;
            }


          } else if (edgeTable[i][1] ==
                edgeTable[j][1] ) {
            if (edgeTable[i][2] >
              edgeTable[j][2]) {
              // swap edge
              for (k = 0; k < 4 ; k++) {
                edgeData = edgeTable[i][k];
                edgeTable[i][k] = edgeTable[j][k];
                edgeTable[j][k] = edgeData;
              }

              // swap planTable
              for (k = 0 ; k < numberOfPhases; k++) {
                tempPlanData =  planTable[k][i];
                planTable[k][i] = planTable[k][j];
                planTable[k][i] = tempPlanData;
              }


            }else if (edgeTable[i][2] ==
                  edgeTable[j][2]) {
              if (edgeTable[i][3] >
                edgeTable[j][3] ) {
                // swap edge
                for (k = 0; k < 4 ; k++) {
                  edgeData = edgeTable[i][k];
                  edgeTable[i][k] = edgeTable[j][k];
                  edgeTable[j][k] = edgeData;
                }

                // swap planTable
                for (k = 0 ; k < numberOfPhases; k++) {
                  tempPlanData =  planTable[k][i];
                  planTable[k][i] = planTable[k][j];
                  planTable[k][i] = tempPlanData;
                }
              }
            }
          }
        }
      }
    }
  }

  public int getAngleOfRotation(){
    return angleOfRotation;
  }

  public void load(String fileName){
    FilePlan filePlan = new FilePlanForOrigami();

    filePlan.read(fileName, this);
  }
}
```

```
package com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigami;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FileObj;
```

```java
import com.drancom.programmableMatter.folding.monitor.MainWindow;
import com.drancom.programmableMatter.folding.monitor.MainWindowForFoldingRobotWiring;
import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.origami.planner.PlannerForWiring;

public class SimulatorForOrigami {

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_airplain\\plan_for_origami_airplain.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_airplain\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 50;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_box\\plan_for_origami_box.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 70;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_sailboat2\\plan_for_origami_sailboat2.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_sailboat2\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 35;

  /**/
    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_bench\\plan_for_origami_save_bench.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_bench\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 70;

  /**/

//    public static final String FILENAME = "c:\\foldingdata\\save_box\\m00070.obj";
//    public static final int NUMBER_OF_FILES= 1;

    Paper[] papers;
    FileObj[] fileObjs;
    MainWindowForFoldingRobotOrigami mainWindow;
    public SimulatorForOrigami() {

    }

    void run() {
      int i;
      String fileName;

      // Initiation
      papers = new Paper[NUMBER_OF_FILES];
      fileObjs = new FileObj[NUMBER_OF_FILES];
      mainWindow = new MainWindowForFoldingRobotOrigami();

      // load
      for (i=0; i < NUMBER_OF_FILES; i++) {
        fileName = String.format(FILENAME, i+1);
        papers[i] = new Paper();
        fileObjs[i] = new FileObj();
        fileObjs[i].load(fileName, papers[i]);
      }

      Planner planer = new PlanerForOrigami(); // new planer

      planer.build(papers);

      planer.exportPlan(PLAN_FILENAME, papers);

      mainWindow.run(papers, (PlanForOrigami) planer.getPlan());

    }

    public static void main(String[] args) {
      // TODO Auto-generated method stub
      SimulatorForOrigami simulator = new SimulatorForOrigami ();
      simulator.run();
    }
}
```

```java
package com.drancom.programmableMatter.folding.simulator.simulatorForOrigami;

import java.awt.List;
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.StringTokenizer;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FilePlan;
import com.drancom.programmableMatter.folding.origami.planner.Plan;
import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring;

public class FilePlanForOrigami implements FilePlan {

  @Override
  public void build(String fileName, Plan plan) {
    int i;
    int j;
    int k;

    int numberOfEdges;
    int numberOfPhases;

    File file = new File(fileName);
    PlanForOrigami planForOrigami = (PlanForOrigami) plan;
    int [][] planTable = planForOrigami.getPlanTable();
    float [][] edgeTable = planForOrigami.getEdgeTable();

    numberOfEdges = planForOrigami.getNumberOfEdges();
    numberOfPhases = planForOrigami.getNumberOfPhases();

    try {
      boolean success = file.createNewFile();
      if (success) {
        // File did not exist and was created
      } else {
      }
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    String bufferLine  = new String();

    try {
      BufferedWriter bufferedWriter = new BufferedWriter (new FileWriter (file));

      // print edges
      bufferLine = String.format("# %d edges \n", numberOfEdges); // "# %d phases\n"
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      bufferLine = String.format("# e startPointX startPointY endPointX endPointX \n", numberOfPhases); // "# %d phases\n"

      // print out to file
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

    for (i = 0; i < numberOfEdges; i++) {
      bufferLine = String.format("e");
      for (j=0 ; j < 4 ; j++) {
        bufferLine += String.format(", %f", edgeTable[i][j]);
      }

      // print out to file

      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

    }

    // print phase
    bufferLine = String.format("# %d phases", numberOfPhases); // "# %d phases\n"
    bufferedWriter.write(bufferLine);
    bufferedWriter.newLine();

    bufferLine = String.format("# p phases planData", numberOfPhases); // "# %d phases\n"

    // print out to file
    bufferedWriter.write(bufferLine);
    bufferedWriter.newLine();

    // planTable [phases][edgeNumber]
    for (i = 0; i < numberOfPhases; i++) {
      for (j = 0; j < numberOfEdges; j++) {
        bufferLine = String.format("p");
        bufferLine += String.format(", %d, %d", i, planTable[i][j]);

        bufferedWriter.write(bufferLine);
        bufferedWriter.newLine();
      }
      // print to file
    }

      // file close
      bufferedWriter.close();

    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
```

```java

  @Override
  public void build(String fileName, Paper[] papers) {
  }

  @SuppressWarnings("deprecation")
  @Override
  public void read(String fileName, Plan plan) {
    PlanForOrigami planForOrigami = (PlanForOrigami) plan;

    int i;
    int j;

    String data;

    int [][] planTable;
    float [][] edgeTable;

    int numberOfPhases;
    int numberOfEdges;

    // Temporally  variable
    ArrayList<int[]> planArrayList = new ArrayList<int[]>();
    ArrayList<float[]> edgeArrayList = new ArrayList<float[]>();

    String head;
    float[] tempEdge;
    int [] tempPlanData;
    int tempPhase;

    File file = new File(fileName);

    numberOfPhases = 0;
    numberOfEdges  = 0;

    try {

      BufferedReader bufferedReader = new BufferedReader(new FileReader(file));
      // read buffer

      while ((data = bufferedReader.readLine()) != null) {

        StringTokenizer st = new StringTokenizer(data, ",");
        if (st.hasMoreElements()) {
          head =  st.nextToken();
          if(head.startsWith("#")) {

          } else if(head.equals("e")) {
            tempEdge = new float[4];
            tempEdge[0] = Float.parseFloat(st.nextToken());
            tempEdge[1] = Float.parseFloat(st.nextToken());
            tempEdge[2] = Float.parseFloat(st.nextToken());
            tempEdge[3] = Float.parseFloat(st.nextToken());
            edgeArrayList.add(tempEdge);

          } else if(head.equals("p")) {
            tempPlanData = new int[2];
            tempPlanData[0] = Integer.parseInt(st.nextToken().trim());
            tempPlanData[1] = Integer.parseInt(st.nextToken().trim());
            planArrayList.add(tempPlanData);
            numberOfPhases++;;
          }
        }
      }

      bufferedReader.close();

    } catch (FileNotFoundException e) {
        e.printStackTrace();

    } catch (IOException e) {
        e.printStackTrace();
    }

    // build plan
    numberOfEdges = edgeArrayList.size();

    edgeTable = new float[numberOfEdges][4];

    for (i=0; i < numberOfEdges ; i++) {
      edgeTable[i][0] = edgeArrayList.get(i)[0];
      edgeTable[i][1] = edgeArrayList.get(i)[1];
      edgeTable[i][2] = edgeArrayList.get(i)[2];
      edgeTable[i][3] = edgeArrayList.get(i)[3];
    }

    numberOfPhases = planArrayList.size() / numberOfEdges;

    planTable = new int [numberOfPhases][numberOfEdges];

    for (i = 0 ; i < numberOfPhases ; i++) {

      for ( j = 0 ; j < numberOfEdges ; j++) {
        tempPlanData = planArrayList.get(j+(i*numberOfEdges));
        if (tempPlanData[0] != i){
          System.out.format("Error: phase error");
        }

        planTable[i][j] = tempPlanData[1];
      }
    }

    planForOrigami.setEdgeTable(edgeTable);
    planForOrigami.setPlanTable(planTable);
    planForOrigami.setNumberOfEdges(numberOfEdges);
    planForOrigami.setNumberOfPhases(numberOfPhases);

  }

  @Override
  public void read(String fileName, Paper[] papers) {
    // TODO Auto-generated method stub
  }

  @Override
  public void build(String fileName, Plan plan, Paper papers[]) {
    // TODO Auto-generated method stub
  }

  @Override
  public void read(String fileName, Plan plan, Paper[] papers) {
    // TODO Auto-generated method stub
  }
}
```

```java
package com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigami;



import java.awt.Frame;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.media.opengl.*;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.Point;
import com.drancom.programmableMatter.folding.controller.paper.Polygon;
import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.drancom.programmableMatter.folding.monitor.MainWindow;
import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
      ;
import com.sun.opengl.util.Animator;


public class MainWindowForFoldingRobotOrigami  extends MainWindow
      implements GLEventListener, MouseListener, MouseMotionListener {

  static final float LINEWIDTH = 3.0f;

  PlanForOrigami plan;
  int [][] planTable;
  int phase;
  int numberOfphases;


  public void run(Paper[] papers, PlanForOrigami plan) {
    super.run(papers);
    this.plan = plan;
    planTable = this.plan.getPlanTable();
    numberOfphases = this.plan.getNumberOfPhases();
    phase = 0;
  }

  public void buildGlPaper(GL gl, Paper paper){
    int i;
    int j;
    int numberOfLine = paper.getNumberOfEdges();
    int numberOfPolygon = paper.getNumberOfPolygons();

    Polygon polygon;
    Point[] polygonPoints;

    Vector startPointVector;
    Vector endPointVector ;

    gl.glShadeModel(GL.GL_FLAT);

    gl.glNormal3f(0.0f, 0.0f, 1.0f);

    /* draw polygon */
    numberOfPolygon = paper.getNumberOfPolygons();

    for (i=0; i<numberOfPolygon; i++) {
      polygon = paper.getPolygon(i);

      polygonPoints = polygon.getPoints();

      gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WRITE, 0);

      gl.glBegin(GL.GL_TRIANGLES);
        for (j=0; j<3; j++){
          gl.glVertex3d(polygonPoints[j].getXOnPaper() * 8
              , polygonPoints[j].getYOnPaper() * 8
              , polygonPoints[j].getZOnPaper() * 8 );

        }
//        for (j=2; j>=0; j--){
//          gl.glVertex3f(polygonPoints[j].getXInReal() * 8
//              , polygonPoints[j].getYInReal() * 8
//              , polygonPoints[j].getZInReal() * 8 );
//
//          }

        gl.glEnd();

      }

      /* draw lines */
      numberOfLine = paper.getNumberOfEdges();

      for (i=0; i<numberOfLine; i++) {

        startPointVector = paper.getLine(i).getStartPoint().getVectorOnPaper
            ();
        endPointVector = paper.getLine(i).getEndPoint().getVectorOnPaper();
        gl.glLineWidth(LINEWIDTH);
      /**/

        if(planTable[0][i] == 1) {
          // actuating
          gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, RED, 0);

        }else if (planTable[0][i] == -1){
          // passive moving
          gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, BLUE, 0);

        }else if (numberOfphases > 1) {
          if(planTable[1][i] == 1) {

            // actuating
            gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, GREEN, 0);

          }else if (planTable[1][i] == -1){

            // passive moving
            gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, YELLOW, 0);

          } else {

            // stop
            gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WRITE, 0);

          }
        } else {

          // stop
          gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WRITE, 0);
        }

        gl.glBegin(GL.GL_LINES);

        gl.glVertex3f((float)startPointVector.getX()*8,
            (float)startPointVector.getY()*8,
            (float)startPointVector.getZ()*8);

        gl.glVertex3f((float)endPointVector.getX()*8,
            (float)endPointVector.getY()*8,
            (float)endPointVector.getZ()*8);
        gl.glEnd();

      }
    }



}
```

```
package com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigamis;

import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.
      MonitorOfPlanGroupOfPlanForOrigamis;
import com.drancom.programmableMatter.folding.origami.planner.Plan;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
      .PlanForOrigami;

public class SimulatorToFindOptimalOrigamisWithInvertingAndRotation {



  public static final String PLAN_FILENAME[] = {
    "V:\\com\\dran\\vc\\pm\\RigidOrigami006\\RigidOrigami\\save_8x8_s-
        shuttle\\plan_for_origami_s-shuttle.csv"
    , "V:\\com\\dran\\vc\\pm\\RigidOrigami006\\RigidOrigami\\save_8x8_hat\\
        plan_for_origami_8x8_hat.csv"
    , "c:\\foldingdata\\save_8x8airplain\\plan_for_origami_8x8airplain.csv"
    , "c:\\foldingdata\\save_8x8sailboat\\plan_for_origami_8x8sailboat.csv"
    , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"
    , "c:\\foldingdata\\save_8x8elephant\\plan_for_origami_save_8x8elephant
        .csv"
    , "c:\\foldingdata\\save_8x8bench\\plan_for_origami_save_8x8bench.csv"
    , "c:\\foldingdata\\save_8x8table\\plan_for_origami_save_8x8table.csv"
    , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"

    };

  public static final int NUMBER_OF_PLAN_FILES= 2;

  public static final String PLAN_FOR_ORIGAMIS_FILENAME = "c:\\foldingdata
      \\save_plan2\\%splan_for_origamis %d.%s";
  public static final String PLAN_FOR_ORIGAMIS_FILETYPE = "csv";

  public SimulatorToFindOptimalOrigamisWithInvertingAndRotation() {
  }

  void run() {
    int i;
    int j;
    int k;

    FileObj[] fileObjs;
    PlanForOrigami[][] plansForOrigami;
    PlanForOrigami[] inputPlansForOrigami;
    MonitorOfPlanGroupOfPlanForOrigamis monitor;

    Planner [] planers;

    int numberOfPlansForOrigamis;
    int optimalPlanForOrigamis;
    int numberOfActiveEdgeOfOptimalPlanForOrigamis;

    int optimalGroupsForOrigamis;
    int numberOfGroupsOfOptimalNumberOfGroups;

    String tempString;

    // Initiation
    fileObjs =  new FileObj[NUMBER_OF_PLAN_FILES];
    plansForOrigami = new PlanForOrigami[NUMBER_OF_PLAN_FILES][8];
    monitor = new MonitorOfPlanGroupOfPlanForOrigamis();

    // loads

    for (i=0; i < NUMBER_OF_PLAN_FILES; i++) {
      for ( j=0; j < 8 ; j ++) {
        plansForOrigami[i][j] = new PlanForOrigami();
        plansForOrigami[i][j].load(PLAN_FILENAME[i]);
      }
    }

    // Optimal Algorithms
    // input : PlansOfOrigami
    // output : planOfOrigamisOfMinimalString;
    //        planOfOrigamisOfMinimalThreading

    // 1. for i=0 to NumberOfPlans
    // 2.       for j=0 to 8
    // 3.            plansOfOrigami[i][j] <- plansOfOrigami[i]
    //
    // 4. for i=0 to NumberOfPlans
    // 5.       for j=0 to 8
    // 6.            if j>=4,
    // 7.          do invert plansOfOrigami[i][j]
    // 8.            for k = 0 to k < j %4
    // 9.            do rotate clockwise plansOfOrigami[i][j]

    for (i=0; i < NUMBER_OF_PLAN_FILES; i++) {
      for ( j=0; j < 8 ; j ++) {
        if (j >= 4 ) {
          plansForOrigami[i][j].invert();
        }
        for (k = 0; k < j % 4; k++) {
          plansForOrigami[i][j].rotateClockwise();
        }
      }
    }

    // 10. numberOfPlansForOrigais <- 1
    // 11. for i=0 to numberOfPlan
    // 12       numberOfPlansForOrigamis *= 8
    numberOfPlansForOrigamis = 1;
    for (i = 0; i < NUMBER_OF_PLAN_FILES; i++) {
      numberOfPlansForOrigamis *= 8;
    }

    // 0, 0 invert, 90, 90 invert 180, 180 invert, 270 invert,  270
    planers = new Planner[numberOfPlansForOrigamis ];
    inputPlansForOrigami = new PlanForOrigami[NUMBER_OF_PLAN_FILES];
    int tempInt;

    // 13. for i = 0 to numberOfPlansForOrigamis
    // 14.       tempInt <- i
    // 15.     for j = 0 to numberOfPlan
    // 16.       inputPlansForOrigami[j] = plansForOrigami[j][tempInt % 8]
    // 17.       tempInt/=8;
    // 18.     planers[i] planAlgoritmForOrigamis(inputPlansForOrigami)

    for (i=0; i < numberOfPlansForOrigamis; i++ ) {
      planers[i] = new PlanerForOrigamis(); // new planer

      tempInt = i;
      for (j = 0; j < NUMBER_OF_PLAN_FILES; j++) {;
        inputPlansForOrigami[j] = plansForOrigami[j][tempInt % 8];
        tempInt /= 8;
      }

      planers[i].build(inputPlansForOrigami);

    }
    // 19. optimalPlanForOrigamis <- 0
    // 20. numberOfActiveEdgeOfOptimalPlanForOrigamis <-
          getNumberOfActiveEdgesInPlanGroup(planers[0])
    optimalPlanForOrigamis = 0;
    numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis) planers
          [0].getPlan()).getNumberOfActiveEdgesInPlanGroup();

    // 21. optimalGroupsForOrigamis <- 0
    // 22. numberOfGroupsOfOptimalNumberOfGroups <- get
          numberOfGroupsOfOptimalNumberOfGroups(planers[0])
    optimalGroupsForOrigamis = 0;
    numberOfGroupsOfOptimalNumberOfGroups = ((PlanForOrigamis) planers[0].
          getPlan()).getNumberOfActivePlanGroup();

    // 23. optimalPlanForOrigamis <- number of index planers having the
          smallest number of Active Edges In PlanGroup
    for (i = 0; i< numberOfPlansForOrigamis; i++) {
      if (        numberOfActiveEdgeOfOptimalPlanForOrigamis >  ((
            PlanForOrigamis) planers[i].getPlan()).
            getNumberOfActiveEdgesInPlanGroup()
        ||  ((numberOfActiveEdgeOfOptimalPlanForOrigamis == ((
             PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlanGroup())
           && (((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan
                ()).getNumberOfActivePlanGroup()
           >   ((PlanForOrigamis) planers[i              ].getPlan()).
                getNumberOfActivePlanGroup()))) {
        optimalPlanForOrigamis = i;
        numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis)
              planers[i].getPlan()).getNumberOfActiveEdgesInPlanGroup();
      }

    // 24. optimalGroupsForOrigamis <- number of index of planers having
          the smallest number of Active PlanGroup
      if (        numberOfGroupsOfOptimalNumberOfGroups >  ((PlanForOrigamis)
              planers[i].getPlan()).getNumberOfActivePlanGroup()
        ||  ((numberOfGroupsOfOptimalNumberOfGroups == ((PlanForOrigamis)
              planers[i].getPlan()).getNumberOfActivePlanGroup())
           && (((PlanForOrigamis) planers[optimalGroupsForOrigamis].
                getPlan()).getNumberOfActiveEdgesInPlanGroup()
           >   ((PlanForOrigamis) planers[i               ].getPlan()).
                getNumberOfActiveEdgesInPlanGroup()))) {
        optimalGroupsForOrigamis = i;
        numberOfGroupsOfOptimalNumberOfGroups = ((PlanForOrigamis) planers[
              i].getPlan()).getNumberOfActivePlanGroup();
      }

      System.out.printf("planForOrigamis \t %d %o %d %d %f %f %d %d ", i, i
          , ((PlanForOrigamis) planers[i].getPlan()).
              getNumberOfActiveEdgesInPlanGroup()
          , ((PlanForOrigamis) planers[i].getPlan()).
              getNumberOfActiveEdgesInPlansForOrigami()
          , (float)((PlanForOrigamis) planers[i].getPlan()).
              getNumberOfActiveEdgesInPlanGroup() / (float) ( (
              PlanForOrigamis) planers[i].getPlan()).
              getNumberOfActiveEdgesInPlansForOrigami()
          , (float)((PlanForOrigamis) planers[i].getPlan()).
              getNumberOfActiveEdgesInPlanGroup() / (float)( ((
              PlanForOrigamis) planers[i].getPlan()).getNumberOfEdges() *
              NUMBER_OF_PLAN_FILES)
          , ((PlanForOrigamis) planers[i].getPlan()).
              getNumberOfActivePlanGroup()
          , ((PlanForOrigamis) planers[i].getPlan()).getNumberOfGroups());

      if(((PlanForOrigamis) planers[i].getPlan()).isAlined()){
        System.out.printf("true\n");
        } else {
          System.out.printf("false\n");
        }
    }

    System.out.printf("optimalEdgePlanForOrigamis \t %d %o %d %d %f %f %d %
          d\n", optimalPlanForOrigamis, optimalPlanForOrigamis
            , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                .getNumberOfActiveEdgesInPlanGroup()
            , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                .getNumberOfActiveEdgesInPlansForOrigami()
            , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
                getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)((
                PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan()
                ).getNumberOfActiveEdgesInPlansForOrigami()
            , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
                getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)
                (((PlanForOrigamis) planers[optimalPlanForOrigamis].
                getPlan()).getNumberOfEdges() * NUMBER_OF_PLAN_FILES)
            , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                .getNumberOfActivePlanGroup()
            , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                .getNumberOfGroups());

    System.out.printf("optimalNumberOfGroups \t %d %o %d %d %f %f %d %d\n",
          optimalGroupsForOrigamis, optimalGroupsForOrigamis
        , ((PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
            getNumberOfActiveEdgesInPlanGroup()
        , ((PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
            getNumberOfActiveEdgesInPlansForOrigami()
        , (float)((PlanForOrigamis) planers[optimalGroupsForOrigamis].
            getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)((
            PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
            getNumberOfActiveEdgesInPlansForOrigami()
        , (float)((PlanForOrigamis) planers[optimalGroupsForOrigamis].
            getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)(((
            PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
            getNumberOfEdges() * NUMBER_OF_PLAN_FILES)
        , ((PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
            getNumberOfActivePlanGroup()
        , ((PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
            getNumberOfGroups());

    planers[optimalPlanForOrigamis].exportPlan(String.format(
          PLAN_FOR_ORIGAMIS_FILENAME, "optimalEdgePlan_",
          NUMBER_OF_PLAN_FILES, PLAN_FOR_ORIGAMIS_FILETYPE));

    planers[optimalGroupsForOrigamis].exportPlan(String.format(
          PLAN_FOR_ORIGAMIS_FILENAME, "optimalNumberOfActiveGroups_",
          NUMBER_OF_PLAN_FILES,PLAN_FOR_ORIGAMIS_FILETYPE));

//    monitor.run((PlanForOrigamis) planers[optimalGroupsForOrigamis].
        getPlan());
//    monitor.run((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan
        ());
//    monitor.run((PlanForOrigamis) planers[8].getPlan());
//    monitor.run((PlanForOrigamis) planers[19].getPlan());

  }

  public static void main(String[] args) {
    SimulatorToFindOptimalOrigamisWithInvertingAndRotation simulator = new
          SimulatorToFindOptimalOrigamisWithInvertingAndRotation ();
    simulator.run();
  }

}
```

```
package com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigamis;

import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.
      MonitorOfPlanGroupOfPlanForOrigamis;
import com.drancom.programmableMatter.folding.origami.planner.Plan;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
      .PlanForOrigami;


public class SimulatorToFindOptimalOrigamisWithInverting {

  public static final String PLAN_FILENAME[] = {
      "c:\\foldingdata\\save_8x8table\\plan_for_origami_save_8x8table.csv"
    , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"
    , "c:\\foldingdata\\save_8x8airplain\\plan_for_origami_8x8airplain.csv"
    , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"
    , "c:\\foldingdata\\save_8x8sailboat\\plan_for_origami_8x8sailboat.csv"
    , "c:\\foldingdata\\save_8x8bench\\plan_for_origami_save_8x8bench.csv"
        };

  public static final int NUMBER_OF_PLAN_FILES= 2;

  public static final String PLAN_FOR_ORIGAMIS_FILENAME = "c:\\foldingdata
        \\save_plan\\plan_for_origamis %d.csv";

    FileObj[] fileObjs;
    Plan[] plans;
    MonitorOfPlanGroupOfPlanForOrigamis monitor;
  public SimulatorToFindOptimalOrigamisWithInverting() {

  }

  void run() {
    int i;
    int j;

    int numberOfPlansForOrigamis;
    int optimalPlanForOrigamis;
    int numberOfActiveEdgeOfOptimalPlanForOrigamis;

    // Initiation
    fileObjs =  new FileObj[NUMBER_OF_PLAN_FILES];
    plans =   new PlanForOrigami[NUMBER_OF_PLAN_FILES];
    monitor = new MonitorOfPlanGroupOfPlanForOrigamis();

    // loads
    for (i=0; i < NUMBER_OF_PLAN_FILES; i++) {
      plans[i] = new PlanForOrigami();
      ((PlanForOrigami) plans[i]).load(PLAN_FILENAME[i]);
    }

    numberOfPlansForOrigamis = 1 << NUMBER_OF_PLAN_FILES;

    // 0, 0 invert, 90, 90 invert 180, 180 invert, 270 invert,  270

    Planner [] planers = new Planner[numberOfPlansForOrigamis ];
    for (i=0; i < numberOfPlansForOrigamis; i++ ) {
      planers[i] = new PlanerForOrigamis(); // new planer

      for (j=0; j < NUMBER_OF_PLAN_FILES; j++) {
        if ( ( i & (1 << j)) != 0 ) {
          if ( ( (PlanForOrigami) plans[j]).isInvert()) {
          } else {
            ( (PlanForOrigami) plans[j]).invert();
          }
        } else {
          if (( (PlanForOrigami) plans[j]).isInvert()) {
            ((PlanForOrigami) plans[j]).invert();
          } else {

          }
        }
      }
      planers[i].build(plans);
      planers[i].exportPlan(String.format(PLAN_FOR_ORIGAMIS_FILENAME, i));
    }
```

```java

    optimalPlanForOrigamis = 0;
    numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis) planers
         [0].getPlan()).getNumberOfActiveEdgesInPlanGroup();
    for (i = 1; i< numberOfPlansForOrigamis; i++) {
      if (numberOfActiveEdgeOfOptimalPlanForOrigamis > ((PlanForOrigamis)
           planers[i].getPlan()).getNumberOfActiveEdgesInPlanGroup()) {
        optimalPlanForOrigamis = i;
        numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis)
             planers[i].getPlan()).getNumberOfActiveEdgesInPlanGroup();
      }
      System.out.printf("planForOrigamis \t %d %d %d %f %f \n", i
          , ((PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlanGroup()
          , ((PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlansForOrigami()
          , (float)((PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlanGroup() / (float) ( (
             PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlansForOrigami()
          , (float)((PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlanGroup() / (float)( ((
             PlanForOrigamis) planers[i].getPlan()).getNumberOfEdges() *
             NUMBER_OF_PLAN_FILES));
    }

    System.out.printf("optimalPlanForOrigamis \t %d %d %d %f %f\n",
         optimalPlanForOrigamis
            , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
               .getNumberOfActiveEdgesInPlanGroup()
            , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
               .getNumberOfActiveEdgesInPlansForOrigami()
            , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
               getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)((
               PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan()
               ).getNumberOfActiveEdgesInPlansForOrigami()
            , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
               getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)
               (((PlanForOrigamis) planers[optimalPlanForOrigamis].
               getPlan()).getNumberOfEdges() * NUMBER_OF_PLAN_FILES));

    //monitor.run((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan
         ());
    monitor.run((PlanForOrigamis) planers[2].getPlan());
  }

  public static void main(String[] args) {
    SimulatorToFindOptimalOrigamisWithInverting simulator = new
         SimulatorToFindOptimalOrigamisWithInverting ();
    simulator.run();
  }
}
```

```java
package com.drancom.programmableMatter.folding.simulator.
     simulatorForOrigamis;

import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.
     MonitorOfPlanGroupOfPlanForOrigamis;
import com.drancom.programmableMatter.folding.origami.planner.Plan;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
     .PlanForOrigami;

public class SimulatorForOrigamis {

  public static final String PLAN_FILENAME[] = {
    "c:\\foldingdata\\save_8x8airplain\\plan_for_origami_8x8airplain.csv"
    , "c:\\foldingdata\\save_8x8sailboat\\plan_for_origami_8x8sailboat.csv"
    , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"
    , "c:\\foldingdata\\save_8x8bench\\plan_for_origami_save_8x8bench.csv"
         };

  public static final int NUMBER_OF_PLAN_FILES= 2;

  public static final String PLAN_FOR_ORIGAMIS_FILENAME = "c:\\foldingdata
       \\save_plan\\plan_for_origamis %d.csv";

  FileObj[] fileObjs;
  Plan[] plans;
  MonitorOfPlanGroupOfPlanForOrigamis monitor;

  public SimulatorForOrigamis() {

  }

  void run() {
    int i;
    int j;

    int numberOfPlansForOrigamis;
    int optimalPlanForOrigamis;
    boolean isThereAlinedOrigami;
    int numberOfActiveEdgeOfOptimalPlanForOrigamis;

    // Initiation
    fileObjs =  new FileObj[NUMBER_OF_PLAN_FILES];
    plans =   new PlanForOrigami[NUMBER_OF_PLAN_FILES];
    monitor = new MonitorOfPlanGroupOfPlanForOrigamis();

    // loads
    for (i=0; i < NUMBER_OF_PLAN_FILES; i++) {
      plans[i] = new PlanForOrigami();
      ((PlanForOrigami) plans[i]).load(PLAN_FILENAME[i]);
    }

    numberOfPlansForOrigamis = 1 << NUMBER_OF_PLAN_FILES;

    // 0, 0 invert, 90, 90 invert 180, 180 invert, 270 invert,  270

    Planner [] planers = new Planner[numberOfPlansForOrigamis ];
    for (i=0; i < numberOfPlansForOrigamis; i++ ) {
      planers[i] = new PlanerForOrigamis(); // new planer

      for (j=0; j < NUMBER_OF_PLAN_FILES; j++) {
        if ( ( i & (1 << j)) != 0 ) {
          if  ( ( (PlanForOrigami) plans[j]).isInvert()) {
          } else {
            ( (PlanForOrigami) plans[j]).invert();
          }
        } else {
          if  ( ( ( PlanForOrigami ) plans[j] ).isInvert()) {
            ( ( PlanForOrigami ) plans[j] ).invert();
          } else {

          }
        }
      }
      planers[i].build(plans);
      planers[i].exportPlan(String.format(PLAN_FOR_ORIGAMIS_FILENAME, i));
    }

    optimalPlanForOrigamis = 0;
    numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis) planers
         [0].getPlan()).getNumberOfActiveEdgesInPlanGroup();

    isThereAlinedOrigami  = false;

    for (i = 1; i< numberOfPlansForOrigamis; i++) {
      if (((PlanForOrigamis) planers[i].getPlan()).isAlined()) {
        isThereAlinedOrigami  = true;
        // finding plan for alined origamis
        if (numberOfActiveEdgeOfOptimalPlanForOrigamis > ((PlanForOrigamis)
              planers[i].getPlan()).getNumberOfActiveEdgesInPlanGroup()) {
          optimalPlanForOrigamis = i;
          numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis)
               planers[i].getPlan()).getNumberOfActiveEdgesInPlanGroup();
        }
      }

      System.out.printf("planForOrigamis \t %d %d %d %f %f ", i
          , ((PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlanGroup()
          , ((PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlansForOrigami()
          , (float) ((PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlanGroup() / (float) ( (
             PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlansForOrigami()
          , (float) ((PlanForOrigamis) planers[i].getPlan()).
             getNumberOfActiveEdgesInPlanGroup() / (float)( ((
             PlanForOrigamis) planers[i].getPlan()).getNumberOfEdges() *
             NUMBER_OF_PLAN_FILES));
      if(((PlanForOrigamis) planers[i].getPlan()).isAlined()){
        System.out.printf("true\n");
        } else {
          System.out.printf("false\n");
        }
    }

    if (isThereAlinedOrigami  == true ) {
      System.out.print("There is alined plen of origami");

      System.out.printf("optimalPlanForOrigamis \t %d %d %d %f %f\n",
            optimalPlanForOrigamis
          , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
             .getNumberOfActiveEdgesInPlanGroup()
          , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
             .getNumberOfActiveEdgesInPlansForOrigami()
          , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
             getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)((
             PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan()
             ).getNumberOfActiveEdgesInPlansForOrigami()
          , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
             getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)
             (((PlanForOrigamis) planers[optimalPlanForOrigamis].
             getPlan()).getNumberOfEdges() * NUMBER_OF_PLAN_FILES));

    } else {
      System.out.print("There is not alined plan origamis  ");
    }

    monitor.run((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan()
        );

    for(i=1;i<Math.pow(2, NUMBER_OF_PLAN_FILES);i++){
      monitor.run((PlanForOrigamis) planers[i].getPlan());
    }
  }

  public static void main(String[] args) {
    SimulatorForOrigamis simulator = new SimulatorForOrigamis ();
    simulator.run();
  }
}
```

```
package com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigamis;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FilePlan;
import com.drancom.programmableMatter.folding.origami.planner.Plan;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
      .PlanForOrigami;

public class FilePlanForOrigamis implements FilePlan {

  @Override
  public void build(String fileName, Plan plan) {
    int i;
    int j;
    int k;
    int l;

    boolean ready;

    int numberOfGroups;
    int numberOfEdges;
    int numberOfOrigamis;
    int numberOfPhases[];

    File file = new File(fileName);
    PlanForOrigamis planForOrigamis = (PlanForOrigamis) plan;

    int [][][]  planTable   = planForOrigamis.getPlanTable();
    int [][]  groupTable  = planForOrigamis.getPlanGroupTable();
    float [][]  edgeTable   = planForOrigamis.getEdgeTable();

    numberOfGroups  = planForOrigamis.getNumberOfGroups();
    numberOfEdges   = planForOrigamis.getNumberOfEdges();
    numberOfOrigamis= planForOrigamis.getNumberOfOrigamis();
    numberOfPhases  = planForOrigamis.getNumberOfPhases();

    try {
      boolean success = file.createNewFile();
      if (success) {
        // File did not exist and was created
      } else {

      }

    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    String bufferLine  = new String();

    try {
      BufferedWriter bufferedWriter = new BufferedWriter (new FileWriter (
            file));

      bufferLine = String.format("# %d activeEdgesInPlansForOrigami",
            planForOrigamis.getNumberOfActiveEdgesInPlansForOrigami());
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      bufferLine = String.format("# %d activeEdgesInPlanGroup",
           planForOrigamis.getNumberOfActiveEdgesInPlanGroup());
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      bufferLine = String.format("# %f activeEdgesInPlanGroup /
           activeEdgesInPlansForOrigami", ((float)planForOrigamis.
           getNumberOfActiveEdgesInPlanGroup() / (float) planForOrigamis.
           getNumberOfActiveEdgesInPlansForOrigami()));
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      bufferLine = String.format("# %f activeEdgesInPlanGroup / (
           numberOfEdges * numberOfGroups)", (float) planForOrigamis.
           getNumberOfActiveEdgesInPlanGroup() / (float) (numberOfEdges *
           numberOfGroups));
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      bufferLine = String.format("# %d edges", numberOfEdges); // "# %d
           phases"
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      bufferLine = String.format("# startPointX startPointY endPointX
           endPointX "); // "# %d phases"
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      // print out to file
      for (i = 0; i <numberOfEdges; i++) {
        bufferLine = String.format("e");
        for (j=0 ; j < 4 ; j++) {
          bufferLine += String.format(", %f", edgeTable[i][j]);
        }

        // print out to file
        bufferedWriter.write(bufferLine);
        bufferedWriter.newLine();
      }

      // print group
      bufferLine = String.format("# %d groups", numberOfGroups); // "# %d
           phases"
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      bufferLine = String.format("# groupNumber groupdata"); // "# %d
           phases"
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      // planGroup [groupNumber][edgeNumber]
      for (i = 0; i < numberOfGroups; i++) {
        for (j = 0; j < numberOfEdges; j++) {
          bufferLine = String.format("g");
          bufferLine += String.format(", %d, %d", i, groupTable[i][j]);

          // print to file
          bufferedWriter.write(bufferLine);
          bufferedWriter.newLine();

        }


      }

      // print origami plan
      bufferLine = String.format("# %d origamis", numberOfOrigamis); // "#
           %d phases"
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();


      // planTable [phases][edgeNumber]
      for (i = 0; i < numberOfOrigamis; i++) {
        bufferLine = String.format("# %d phases", numberOfPhases[i]); // "#
             %d phases"
        bufferedWriter.write(bufferLine);
        bufferedWriter.newLine();

        bufferLine = String.format("# origamis phases planData"); // "# %d
             phases"
        bufferedWriter.write(bufferLine);
        bufferedWriter.newLine();

        for (j = 0; j < numberOfPhases[i]; j++) {
          for (k = 0; k < numberOfGroups; k++) {
            bufferLine = String.format("p");
            bufferLine += String.format(", %d, %d, %d", i, j, planTable[i][
                 j][k]);


            // print to file
            bufferedWriter.write(bufferLine);
            bufferedWriter.newLine();

          }
        }
      }

      // file close

      bufferedWriter.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  @Override
  public void build(String fileName, Paper[] papers) {
    // TODO Auto-generated method stub

  }

  @Override
  public void read(String fileName, Plan plan) {
  }

  @Override
  public void build(String fileName, Plan plan, Paper[] papers) {

  }

  @Override
  public void read(String fileName, Paper[] papers) {
    // TODO Auto-generated method stub

  }

  @Override
  public void read(String fileName, Plan plan, Paper[] papers) {
    // TODO Auto-generated method stub

  }
}
```

```
package com.drancom.programmableMatter.folding.simulator.
    simulatorForOrigamis;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FilePlan;
import com.drancom.programmableMatter.folding.origami.planner.Plan;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
    .PlanForOrigami;

public class PlanerForOrigamis implements Planner {


  PlanForOrigamis planForOrigamis;

  @Override
  public void build(Paper[] papers) {

  }

  public void build(Plan[] plans) {


//      PlanForOrigamis()
//      Description: Plan for many of Origamis on one Folding Robot with
    minimal loops of SMA, by grouping folding edges for common angles for
    origamis inputted
//      Default:
//      Input:      PlansForOrigami[numberOfOrigamis]

    PlanForOrigami [] plansForOrigami = (PlanForOrigami[]) plans;

    int i;
    int j;
    int k;
    int l;

    int numberOfOrigamis = plansForOrigami.length;
    int [] lastPhase = new int[numberOfOrigamis];

    int maxPhase = 0;
    int numberOfEdges = plansForOrigami[0].getNumberOfEdges();

    int numberOfTables;
    int numberOfGroups;
    int numberOfActiveEdgesInPlansForOrigami;

    numberOfTables = 0;
    for (i = 0; i < numberOfOrigamis; i++ ){
      lastPhase[i] = plansForOrigami[i].getNumberOfPhases();
      if (maxPhase < lastPhase[i]) {
        maxPhase = lastPhase[i];
      }
      if (numberOfEdges != plansForOrigami[i].getNumberOfEdges()) {
        System.out.print("Error: Number Of Edge");
      }
      numberOfTables += lastPhase[i];
    }

    int planForOrigami[][][] = new int [numberOfOrigamis][maxPhase][
        numberOfEdges];

    for (i = 0; i < numberOfOrigamis; i++) {
      for (j=0; j < lastPhase[i]; j++) {
        for (k = 0 ; k < numberOfEdges; k++) {
          planForOrigami[i][j][k] = plansForOrigami[i].getPlanTable()[j][k
              ];
        }
      }
    }


//      Output:
//      planTableForOrigamis
//      planGroup

    int[][]      planGroup ;
    int[][][] planTableForOrigamis;

    int setOfNumber;
    int detector;
    int sumOfTable;
//    A. Extract a ThreadingTable from PlansForOrigami.
//         PlansForOrigami has a one or more edge information of folding.
//       ThreadingTable is 2D Array
//    1. NumberOfTables <- 0
//    2. for i=1 to numberOfOrigamis
//    3.           numberOfTables += lastPhase[i]
    numberOfTables = 0;
    for (i = 0; i < numberOfOrigamis; i++ ){
      numberOfTables += lastPhase[i];
    }

//       4.  tableNumber <- 0
//    5.  For i=1 to numberOfOrigamis
//    6.           For j=0 to lastPhase[i]
//    7.                   ThreadingTable[NumberOfTable][1..n] =
   PlanForOrigami[i][j][1..n]
//    8.                   tableNumber++;
    int[][] threadingTable = new int [numberOfTables][numberOfEdges];

    int tableNumber = 0;
    for (i=0; i < numberOfOrigamis; i++){
      for (j = 0; j < lastPhase[i] ; j++) {
        for (k=0; k < numberOfEdges; k++) {
          threadingTable [tableNumber][k] = planForOrigami[i][j][k];

        }
        tableNumber++;
      }
    }

//      9.  numberOfGroups = 0
    numberOfGroups = 0;

//      10. For i = 0 to numberOfTable -1
//      11.          numberOfGroups +=  2  ^  i
    for (i=0 ; i < numberOfTables ; i++) {
      numberOfGroups += (1 << i);
    }

//      12. PlanGroup[1..numberOfGroups][1..numberOfEdges]=0
    planGroup = new int[numberOfGroups][numberOfEdges];
    for (i=0; i< numberOfGroups; i++) {
      for (j=0; j<numberOfEdges; j++) {
        planGroup[i][j] = 0;
      }
    }


//      13. For i = 1 to numberOfEdges
//      14.    groupNumberForMount <- 0
//      15.    groupNumberForVally <- 0
//      16.      For j = 0 to numberOfTables - 1
//      17.      if threadingTable[j][i] == 1,
//      18.            do groupNumberForMount += (2 ^ j) * threadingTable[j][i
    ];
//        19.        if threadingTable[j][i] == -1,
//        20.                 do groupNumberForVally += (2 ^ j) * threadingTable[
      j][i];
//        21.        if groupNumberForMount > 0,
//        22.        do planGroup[groupNumberForMount - 1] = 1
//      23.        if groupNumberForVally > 0,
//        24.        do planGroup[groupNumberForVally - 1] = -1

    int groupNumberForMount;
    int groupNumberForVally;
    for (i = 0 ; i < numberOfEdges; i++) {
      groupNumberForMount = 0;
      groupNumberForVally = 0;

      for (j = 0; j < numberOfTables; j++) {
        if (threadingTable[j][i] == 1) {
          groupNumberForMount += (1 << j);
        }
        if (threadingTable[j][i] == -1) {
          groupNumberForVally += (1 << j);
        }
      }
      if (groupNumberForMount > 0) {
        planGroup[groupNumberForMount - 1][i] = 1;
      }
      if (groupNumberForVally > 0) {
        planGroup[groupNumberForVally - 1][i] = -1;
      }
    }

//      25. PlanTableForOrigamis[1..numberOfOrigamis][1..Max(lastPhase[1..])
      ][1..numberOfGroups] <- 0

    planTableForOrigamis = new int [numberOfOrigamis][maxPhase][
        numberOfGroups];

    for (i=0; i<numberOfOrigamis; i++) {
      for (j=0; j < maxPhase; j++ ) {
        for (k = 0; k < numberOfGroups; k++) {
          planTableForOrigamis[i][j][k] = 0;
        }
      }
    }

//      26. For i = 1 to numberOfOrigamis
//      27.          For j = 0 to lastPhase[i]
//      28.                  For k = 1 to numberOfEdge
//      29.                  If PlanForOrigami[i][j][k] != 0
//      30.                            Do For L = 1 to numberOfGroups
//      31.                                         If PlanGroup[L][k] ==
      PlanForOrigami[i][j][k]
//      32.                                                       Do PlanForOrigamis[i][j][
      k][L] <- 1

    for (i = 0; i < numberOfOrigamis; i++) {
      for (j = 0 ; j < lastPhase[i] ; j++) {
        for (k = 0; k < numberOfEdges; k++) {
          if (planForOrigami[i][j][k] != 0) {
            for (l = 0 ; l < numberOfGroups ; l++) {
              if (planGroup[l][k] == planForOrigami[i][j][k]){
                planTableForOrigamis[i][j][l] = 1;
              }
            }
          }
        }
      }
    }

// number Of Active Edges In Plans For Origami
    numberOfActiveEdgesInPlansForOrigami = 0;
    for(i=0; i < numberOfTables; i++) {
      for(j=0; j<numberOfEdges; j++) {
        if (threadingTable[i][j] != 0){
          numberOfActiveEdgesInPlansForOrigami++;
        }
      }
    }

//  Finding Optimal Electronic Power Input Vertexes Algorithm
    // only for 8x8
    int x0, y0, x1, y1;
    float edgeTable[][] = plansForOrigami[0].getEdgeTable();
    float vertexs[][][][] = new float[numberOfGroups][9][9][4]; // [
        numberOfGroups][x][y][0:x position , 1: y position , 2:
        NumberOfConnectedActiveEdges , 3:tag]

    int searchMode = 0;// 0 = looking for +, 1 = looking for = -

    for (i = 0; i< numberOfGroups; i++) {
      for (j = 0; j < 9; j++) {
        for (k = 0; k < 9; k++) {
          vertexs[i][j][k][0] = (1.0f / 8.0f) * (float) j;
          vertexs[i][j][k][1] = (1.0f / 8.0f) * (float)k;
          vertexs[i][j][k][2] = 0;
          vertexs[i][j][k][3] = 4;
        }
      }
      for (j = 0 ; j < numberOfEdges ; j++) {
        if( planGroup[i][j] != 0 ) {

          x0 = (int)((edgeTable[j][0])* 8);
          y0 = (int)(edgeTable[j][1]* 8);
          x1 = (int)((edgeTable[j][2])* 8);
          y1 = (int)(edgeTable[j][3]* 8);

          if( x0 <0 ) {
            x0 = (int)((edgeTable[j][0]+1)* 8);
          }
          if( x1 <0 ) {
            x1 = (int)((edgeTable[j][2]+1)* 8);
          }

          vertexs[i][x0][y0][2]++;
          vertexs[i][x0][y0][3] = 0;
          vertexs[i][x1][y1][2]++;
          vertexs[i][x1][y1][3] = 0;

        }
      }
    }



    // tag 1 = +
    // tag 2 = -
    // tag 3 = connected
    // tag 4 = not connected
    boolean isNo0Tag;
    boolean isNoMoreEdgeToTag;

    isNoMoreEdgeToTag = true;
    for (i = 0; i< numberOfGroups; i++) {
      isNo0Tag = true;
      for (j=0; j < numberOfEdges; j++) {
        if (planGroup[i][j] != 0) {
          isNo0Tag = false;
          break;
        }
      }
    searchMode = 0;
    while (!isNo0Tag) {
      if(isNoMoreEdgeToTag == true){
        searchMode = 0;
      }

      isNoMoreEdgeToTag = true;
      for (j=0;j<numberOfEdges;j++) {
        if(planGroup[i][j] != 0) {

          x0 = (int)((edgeTable[j][0])* 8);
          y0 = (int)(edgeTable[j][1]* 8);
          x1 = (int)((edgeTable[j][2])* 8);
          y1 = (int)(edgeTable[j][3]* 8);

          if( x0 <0 ) {
            x0 = (int)((edgeTable[j][0]+1)* 8);
          }
          if( x1 <0 ) {
            x1 = (int)((edgeTable[j][2]+1)* 8);
          }
          if (searchMode == 0 ){
            if(vertexs[i][x0][y0][3] == 0
                && vertexs[i][x0][y0][2] == 1
                && vertexs[i][x1][y1][3] == 0
                && vertexs[i][x1][y1][2] == 1 ){
              vertexs[i][x0][y0][3] = 1;
              vertexs[i][x1][y1][3] = 2;


            } else if(vertexs[i][x0][y0][3] == 0
                && vertexs[i][x0][y0][2] == 1
                && vertexs[i][x1][y1][3] == 0
                && vertexs[i][x1][y1][2] > 1 ){
              vertexs[i][x0][y0][3] = 1;
              vertexs[i][x1][y1][3] = 3;

              searchMode = 1;
              isNoMoreEdgeToTag = false;

            } else if(vertexs[i][x0][y0][3] == 0
                && vertexs[i][x0][y0][2] > 1
                && vertexs[i][x1][y1][3] == 0
                && vertexs[i][x1][y1][2] == 1 ){
              vertexs[i][x0][y0][3] = 3;
              vertexs[i][x1][y1][3] = 1;

              searchMode = 1;
              isNoMoreEdgeToTag = false;

            }
          }else if ( searchMode == 1) {

            if(vertexs[i][x0][y0][3] == 3
              && vertexs[i][x0][y0][2] > 1
              && vertexs[i][x1][y1][3] == 0
              && vertexs[i][x1][y1][2] > 1 ){
              vertexs[i][x1][y1][3] = 3;
              isNoMoreEdgeToTag = false;

            } else if(vertexs[i][x0][y0][3] == 0
                && vertexs[i][x0][y0][2] > 1
                && vertexs[i][x1][y1][3] == 3
                && vertexs[i][x1][y1][2] > 1 ){
              vertexs[i][x0][y0][3] = 3;
              isNoMoreEdgeToTag = false;

            } else if(vertexs[i][x0][y0][3] == 3
                && vertexs[i][x0][y0][2] >= 1
                && vertexs[i][x1][y1][3] == 0
                && vertexs[i][x1][y1][2] == 1 ){
              vertexs[i][x1][y1][3] = 2;

              isNoMoreEdgeToTag = false;

            } else if(vertexs[i][x0][y0][3] == 0
                && vertexs[i][x0][y0][2] == 1
                && vertexs[i][x1][y1][3] == 3
                && vertexs[i][x1][y1][2] >= 1 ){
              vertexs[i][x0][y0][3] = 2;
              isNoMoreEdgeToTag = false;


            }
          }
        }
      }
      if(isNoMoreEdgeToTag == true && searchMode == 0){
        for (j=0;j<numberOfEdges;j++) {
          if(planGroup[i][j] != 0) {
            if(planGroup[i][j] != 0) {


              x0 = (int)((edgeTable[j][0])* 8);
              y0 = (int)(edgeTable[j][1]* 8);
              x1 = (int)((edgeTable[j][2])* 8);
              y1 = (int)(edgeTable[j][3]* 8);


              if( x0 <0 ) {
                x0 = (int)((edgeTable[j][0]+1)* 8);
              }
              if( x1 <0 ) {
                x1 = (int)((edgeTable[j][2]+1)* 8);
              }

              if(searchMode == 0
                && vertexs[i][x0][y0][3] == 0
                && vertexs[i][x0][y0][2] >= 1
                && vertexs[i][x1][y1][3] == 0
                && vertexs[i][x1][y1][2] >= 1 ){

                vertexs[i][x0][y0][3] = 2;
                vertexs[i][x1][y1][3] = 3;
                isNoMoreEdgeToTag = false;

                searchMode = 1;
              }
            }
          }
        }
      }

      isNo0Tag = true;
      for (j = 0; j < 9; j++) {
        for (k = 0; k < 9; k++) {
          if (vertexs[i][j][k][3] == 0){
            isNo0Tag = false;
            break;
          }
        }
      }
    }
  }



  // 1. find a vertex connected nothing active edge. put tag 4 on it
```

```
    // 2. find a vertex connected one active edge. put tag 1 on it.
    // 3. find a vertex connected tag 1 or 3 vertex through active edge .
    //    if it is not a dead end put tag 3 on. if it is dead end put tag 2
    // 4. go to 2
    // 5. go to 1



    // build plan
    planForOrigamis = new PlanForOrigamis();

    planForOrigamis.setPlanTable(planTableForOrigamis);
    planForOrigamis.setPlanGroupTable(planGroup);
    planForOrigamis.setEdgeTable(plansForOrigami[0].getEdgeTable());

    planForOrigamis.setNumberOfOrigamis(numberOfOrigamis);
    planForOrigamis.setNumberOfEdges(numberOfEdges);
    planForOrigamis.setNumberOfPhases(lastPhase);
    planForOrigamis.setNumberOfGroups(numberOfGroups);

    planForOrigamis.setNumberOfActiveEdgesInPlansForOrigami(
          numberOfActiveEdgesInPlansForOrigami);

    // Vertex
    planForOrigamis.setVertexs(vertexs);
  }

  @Override
  public void exportPlan(String fileName) {
    planForOrigamis.export(fileName);
  }

  @Override
  public void exportPlan(String fileName, Paper[] papers) {
    FilePlan filePlan = new FilePlanForOrigamis();

    filePlan.build(fileName, planForOrigamis);
  }

  @Override
  public Plan getPlan() {
    return this.planForOrigamis;
  }

}
```

```
package com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigamis;

import com.drancom.programmableMatter.folding.dataFile.FilePlan;
import com.drancom.programmableMatter.folding.origami.planner.Plan;

public class PlanForOrigamis implements Plan {
  float edgeTable[][];  // [edge number] [x0 y0 x1 y2]
  int planTable[][][];  // [numberOfOrigamis] [numberOfPhases] [
        numberOfGroups]
  int planGroup[][];     // [numberOfGroups] [numberOfEdges]

  float vertexs[][][][]; // [numberOfGroups][x][y][0:x position , 1: y
        position , 2: NumberOfConnectedActiveEdges , 3:tag]

  int numberOfEdges;
  int numberOfPhases[];
  int numberOfGroups;
  int numberOfOrigamis;
  int numberOfActiveEdgesInPlansForOrigami;

  public boolean isAlined() {

    int i;
    int j;

    boolean isFoldingOneway;

    for (i = 0; i < getNumberOfEdges(); i++) {
      isFoldingOneway = false;
      for (j = 0; j < getNumberOfGroups(); j++) {
        if(planGroup[j][i] != 0) {
          if (isFoldingOneway == true){
            return false;
          } else {
            isFoldingOneway = true;
          }
        }
      }
    }
    return true;
  }

  public int [][] getConbinedPlanMap() {
    int i;
    int j;

    int [][] conbinedPlanMap = new int[numberOfGroups][numberOfEdges];

    for ( i = 0 ; i < numberOfEdges ; i++) {
      for (j = 0 ; j < numberOfGroups ; j++) {
        conbinedPlanMap[j][i] = 0;
      }
    }


    for ( i = 0 ; i < numberOfEdges ; i++) {
      for (j = 0 ; j < numberOfGroups ; j++) {
        if (planGroup[j][i] == 1) {
          if (conbinedPlanMap[j][i] != 0) {
            conbinedPlanMap[j][i] = 1;
          } else {
            conbinedPlanMap[j][i] = 2;
          }
        } else if (planGroup[j][i] == -1) {
          if (conbinedPlanMap[j][i] != 0) {
            conbinedPlanMap[j][i] = 1;
          } else {
            conbinedPlanMap[j][i] = -1;
          }
        }
      }
    }


    return conbinedPlanMap;

  }

  public void setNumberOfOrigamis(int numberOfOrigamis) {
    this.numberOfOrigamis = numberOfOrigamis;
  }

  public int getNumberOfOrigamis(){
    return numberOfOrigamis;
  }

  public int[][][] getPlanTable(){
    return planTable;
  }

  public void setPlanTable(int[][][] planTable){
    int i;
    int j;
    int k;

    this.planTable = new int [planTable.length][planTable[0].length][
         planTable[0][0].length];

    for (i=0; i<planTable.length ; i++) {
      for(j=0; j< planTable[i].length ; j++) {
        for(k=0; k< planTable[i][j].length ; k++) {
          this.planTable[i][j][k] = planTable[i][j][k];
        }
      }
    }
  }

  public int[][] getPlanGroupTable(){
    return planGroup;
  }

  public void setPlanGroupTable(int [][] planGroup) {
    int i;
    int j;

    this.planGroup = new int [planGroup.length][planGroup[0].length];

    for (i=0; i<planGroup.length ; i++) {
      for(j=0; j< planGroup[i].length ; j++) {
        this.planGroup[i][j] = planGroup[i][j];
      }
    }
  }

  public void setEdgeTable(float [][] edgeTable){
    int i;
    int j;

    this.edgeTable = new float [edgeTable.length][edgeTable[0].length];

    for (i=0; i<edgeTable.length ; i++) {
      for(j=0; j< edgeTable[i].length ; j++) {
        this.edgeTable[i][j] = edgeTable[i][j];
      }
    }
  }

  public float[][] getEdgeTable(){
    return edgeTable;
  }

  public float[][][][] getVertexs() {
    return vertexs;
  }

  public void setVertexs(float[][][][] vertexs) {
    this.vertexs = vertexs;
  }

  public int[] getNumberOfPhases() {
    return numberOfPhases;
  }

  public void setNumberOfPhases(int[] numberOfPhases) {
    int i;

    this.numberOfPhases = new int [numberOfPhases.length];

    for (i=0; i<numberOfPhases.length ; i++) {
      this.numberOfPhases[i] = numberOfPhases[i];
```

```
    }
  }

  public int getNumberOfEdges() {
    return numberOfEdges;
  }

  public void setNumberOfEdges(int numberOfEdges) {
    this.numberOfEdges = numberOfEdges;
  }

  public void export (String fileName) {
    FilePlan filePlan = new FilePlanForOrigamis();
    filePlan.build(fileName, this);
  }

  public void load(String fileName){
    FilePlan filePlan = new FilePlanForOrigamis();
    filePlan.read(fileName, this);
  }

  public int getNumberOfGroups() {
    return numberOfGroups;
  }

  public void setNumberOfGroups(int numberOfGroups) {
    this.numberOfGroups = numberOfGroups;
  }

  public int getNumberOfActiveEdgesInPlanGroup() {
    int i;
    int j;

    int numberOfActiveEdges = 0;

    for (i=0; i < numberOfGroups ; i++) {
      for (j=0; j < numberOfEdges; j++) {
        if (planGroup[i][j] != 0) {
          numberOfActiveEdges++;
        }
      }
    }
    return numberOfActiveEdges;
  }

  public int getNumberOfActivePlanGroup() {
    int i;
    int j;

    int numberOfActivePlanGroup = 0;

    for (i = 0 ; i < numberOfGroups ; i++) {
      for (j = 0 ; j < numberOfEdges ; j++) {
        if (planGroup[i][j] != 0) {
          numberOfActivePlanGroup++;
          break;
        }
      }
    }
    return numberOfActivePlanGroup;
  }

  public void setNumberOfActiveEdgesInPlansForOrigami(int
       numberOfActiveEdgesInPlansForOrigami) {
    this.numberOfActiveEdgesInPlansForOrigami =
         numberOfActiveEdgesInPlansForOrigami;
  }

  public int getNumberOfActiveEdgesInPlansForOrigami() {
    return numberOfActiveEdgesInPlansForOrigami;
  }
}
```

```
package com.drancom.programmableMatter.folding.controller.paper;

import java.awt.List;
import java.util.ArrayList;

import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.drancom.programmableMatter.folding.monitor.MonitorOfPaperArray;
import com.sun.corba.se.spi.legacy.connection.GetEndPointInfoAgainException
    ;
import com.sun.org.apache.bcel.internal.generic.NEWARRAY;
import com.sun.xml.internal.bind.marshaller.MinimumEscapeHandler;

public class Paper {

  ArrayList<Paper> papersForMonitor;
  MonitorOfPaperArray monitor = new MonitorOfPaperArray();

  public final static int MAX_NUMBER_OF_POINTS = 1000;
  public final static int MAX_NUMBER_OF_POLYGONS = 1000;
  public final static int MAX_NUMBER_OF_LINES = 1000;

  public final static double ERROR_RATIO_FOR_LANGTH = 0.05f;
  public final static double MAX_NUMBER_OF_LINKAGE_EFFECT = 0.01f;

  Point[] pointsOnPaper;
  int numberOfPoints = 0;
  Line[] lines;
  protected int numberOfLines = 0;
  Polygon[] polygons;
  int numberOfPolygons;

  Point sortedPointsMetrix[][];
  Point sortedPointsLine[];

  double barEnergyOfPaper = 0.0f;

  public Paper() {
    pointsOnPaper = new Point[MAX_NUMBER_OF_POINTS];
    lines = new Line[MAX_NUMBER_OF_LINES];
    polygons = new Polygon[MAX_NUMBER_OF_POLYGONS];
    // numberOfPoints;
    // numberOfLines;
  }

  // get the values
  public Point getPoint(int index) {
    return pointsOnPaper[index];
  }

  public Point getPoint(double x, double y, double z) {
    int i;
    for (i = 0; i < getNumberOfPoints(); i++) {
      if (pointsOnPaper[i].getXOnPaper() == x && pointsOnPaper[i].
          getYOnPaper() == y
          && pointsOnPaper[i].getZOnPaper() == z) {
        return pointsOnPaper[i];

      }
    }
    return null;
  }

  public Point[] getPoints() {
    return pointsOnPaper;
  }

  public Line[] getLines() {
    return lines;
  }

  public Polygon[] getPolygons() {
    return polygons;
  }

  public Line getLine(int index) {
    return lines[index];
  }

  public int getNumberOfPolygons() {
    return numberOfPolygons;
  }

  public int getNumberOfColoums() {
    return (int) Math.sqrt((double) numberOfPoints);
  }

  // set the values
  public void setPoint(int index, Point point) {
    if (index > MAX_NUMBER_OF_POINTS) {
      System.err.print("over MAX_NUMBER_OF_POINTS");
    }

    this.pointsOnPaper[index] = point;

    if (index > numberOfPoints - 1) {
      numberOfPoints = index + 1;
    }
  }

  public void setLine(int index, Line line) {
    if (index > MAX_NUMBER_OF_LINES) {
      System.err.printf("over MAX_NUMBER_OF_LINES");
    }

    this.lines[index] = line;

    if (index > numberOfLines - 1) {
      numberOfLines = index + 1;
    }
  }

  // build paper
  public void build() {
    // reset
    copyFromCoordinationOnPaperToCoordinationInReal();

    // Sorting Points
    buildSortedPoints();

    // set up Groups of Poligons
    buildPolygons();

    buildPointsByAnglesOfLines();
  }

  // sort line
  public void sortLine() {
    Line sortedLines[] = new Line[numberOfLines];
    Line tempLine;

    int i;
    int j;

    for (i = 0; i < numberOfLines; i++) {
      sortedLines[i] = lines[i];
    }

    for (i = 0; i < numberOfLines - 1; i++) {
      for (j = i + 1; j < numberOfLines; j++) {

        if (sortedLines[i].getStartPoint().getXOnPaper() > sortedLines[j]
            .getStartPoint().getXOnPaper()) {
          // swap
          tempLine = sortedLines[i];
          sortedLines[i] = sortedLines[j];
          sortedLines[j] = tempLine;

        } else if (sortedLines[i].getStartPoint().getXOnPaper() ==
              sortedLines[j]
            .getStartPoint().getXOnPaper()) {
          if (sortedLines[i].getStartPoint().getYOnPaper() > sortedLines[j]
              .getStartPoint().getYOnPaper()) {
            // swap
            tempLine = sortedLines[i];
            sortedLines[i] = sortedLines[j];
            sortedLines[j] = tempLine;

          } else if (sortedLines[i].getStartPoint().getYOnPaper() ==
                sortedLines[j]
              .getStartPoint().getYOnPaper()) {
            if (sortedLines[i].getEndPoint().getXOnPaper() > sortedLines[j]
                .getEndPoint().getXOnPaper()) {
              // swap
              tempLine = sortedLines[i];
              sortedLines[i] = sortedLines[j];
              sortedLines[j] = tempLine;

            } else if (sortedLines[i].getEndPoint().getXOnPaper() ==
                  sortedLines[j]
                .getEndPoint().getXOnPaper()) {
              if (sortedLines[i].getEndPoint().getYOnPaper() > sortedLines[
                  j]
                  .getEndPoint().getYOnPaper()) {

                // swap
                tempLine = sortedLines[i];
                sortedLines[i] = sortedLines[j];
                sortedLines[j] = tempLine;

              }
            }
          }
        }
      }
    }
    lines = sortedLines;

  }

  // point reset
  void copyFromCoordinationOnPaperToCoordinationInReal() {
    int i;

    for (i = 0; i < numberOfPoints; i++) {
      pointsOnPaper[i].setPointInReal(pointsOnPaper[i].getXOnPaper(),
            pointsOnPaper[i]
          .getYOnPaper(), pointsOnPaper[i].getZOnPaper());
      pointsOnPaper[i].isRenewed = false;
    }
  }

  public void buildPointsByAnglesOfLines() {
    int i, j, k, l;
    int numberOfColumes = (int) Math.sqrt((double) numberOfPoints);

    int numberOfRenewedPoint;

    Vector originVector;

    int counterpartColume;

    Point counterpartPoint;
    Point axisLinePoint1;
    Point axisLinePoint2;
    Point originPoint;

    for (i = 0; i < numberOfPoints; i++) {
      pointsOnPaper[i].isRenewed = false;
    }

    // fold by angle
    numberOfRenewedPoint = 0;
    sortedPointsMetrix[0][0].isRenewed = true;
    sortedPointsMetrix[1][1].isRenewed = true;
    sortedPointsMetrix[1][0].isRenewed = true;
    numberOfRenewedPoint = 3;

    i = 0;
    j = 0;
    k = 0;
    l = 0;

    while (!(numberOfRenewedPoint >= numberOfPoints)) {
      for (i = 0; i < numberOfColumes; i++) {
        for (j = 0; j < numberOfColumes; j++) {
          if (!sortedPointsMetrix[i][j].isRenewed()) {
            if (sortedPointsMetrix[i][j].getType() == Point.
                TYPE_ONLY_STRIGHT_LINE_CLOSS) {
              for (k = i - 1; k <= i + 1; k += 2) {
                for (l = j - 1; l <= j + 1; l += 2) {
                  if ((!sortedPointsMetrix[i][j].isRenewed())
                      && (k >= 0 && k < numberOfColumes
                          && l >= 0 && l < numberOfColumes)) {
                    if (sortedPointsMetrix[k][l]
                        .isRenewed()
                        && sortedPointsMetrix[k][j]
                            .isRenewed()
                        && sortedPointsMetrix[i][l]
                            .isRenewed()) {

                      counterpartPoint = sortedPointsMetrix[k][l];
                      if ((k < i && l < j)
                          || (k > i && l > j)) {
                        axisLinePoint1 = sortedPointsMetrix[i][l];
                        axisLinePoint2 = sortedPointsMetrix[k][j];
                      } else { // if (( k < i && l > j )
                        // || (k > i && l < j))
                        // {
                        axisLinePoint1 = sortedPointsMetrix[k][j];
                        axisLinePoint2 = sortedPointsMetrix[i][l];
                      }

                      // get originVector
                      originVector = new Vector();
                      originVector
                          .setXYZ(
                              ((axisLinePoint1
                                  .getXInReal() + axisLinePoint2
                                  .getXInReal()) / 2),
                              ((axisLinePoint1
                                  .getYInReal() + axisLinePoint2
                                  .getYInReal()) / 2),
                            ((axisLinePoint1
                                .getZInReal() + axisLinePoint2
                                .getZInReal()) / 2));

                    if (buildPointInRealByAngle(
                        sortedPointsMetrix[i][j],
                        counterpartPoint,
                        originVector,
                        axisLinePoint1,
                        axisLinePoint2)) {

                      numberOfRenewedPoint++;
                    }
                  }
                }
              }
            }
          } else if (sortedPointsMetrix[i][j].getType() == Point.
            TYPE_OBLIQUE_LINE_CLOSS) {

            for (k = i - 1; k <= i + 1; k += 2) {
              for (l = j - 1; l <= j + 1; l += 2) {
                if ((!sortedPointsMetrix[i][j].isRenewed())
                    && (k >= 0 && k < numberOfColumes
                        && l >= 0 && l < numberOfColumes)) {
                  if (sortedPointsMetrix[k][l]
                      .isRenewed()) {
                    if (sortedPointsMetrix[k][j]
                        .isRenewed()) {
                      counterpartColume = (k - i) * 2
                          + i;
                      if ((counterpartColume >= 0)
                          && (counterpartColume < numberOfColumes)) {
                        if (sortedPointsMetrix[counterpartColume][j]
                            .isRenewed()) {

                          counterpartPoint = sortedPointsMetrix[
                               counterpartColume][j];
                          originPoint = sortedPointsMetrix[k][j];

                          if ((counterpartColume < i && l < j)
                              || (counterpartColume > i && l > j)) {
                            axisLinePoint1 = sortedPointsMetrix[k][l];
                            axisLinePoint2 = sortedPointsMetrix[k][j];
                          } else { // if ((
                            // counterpartColume
                            // < i && l
                            // > j) || (
                            // counterpartColume
                            // > i && l
                            // < j) ) {
                            axisLinePoint1 = sortedPointsMetrix[k][j];
                            axisLinePoint2 = sortedPointsMetrix[k][l];
                          }

                          originVector = sortedPointsMetrix[k][j]
                              .getVectorInReal();

                          if (buildPointInRealByAngle(
                              sortedPointsMetrix[i][j],
                              counterpartPoint,
                              originVector,
                              axisLinePoint1,
                              axisLinePoint2)) {
                            // isRenewed
                            numberOfRenewedPoint++;
                          }
                        }
                      }
                    } else if (sortedPointsMetrix[i][l]
                        .isRenewed()) {
                      counterpartColume = (l - j) * 2
                          + j;
                      if ((counterpartColume >= 0)
                          && (counterpartColume < numberOfColumes)) {
                        if (sortedPointsMetrix[i][counterpartColume]
                            .isRenewed()) {
                          counterpartPoint = sortedPointsMetrix[i][
                               counterpartColume];
                          originPoint = sortedPointsMetrix[i][l];
                          if ((k < i && counterpartColume < j)
                              || (k > i && counterpartColume > j)) {
                            axisLinePoint1 = sortedPointsMetrix[i][l];
                            axisLinePoint2 = sortedPointsMetrix[k][l];
                          } else {// if ((k > i &&
                            // counterpartColume
                            // < j) || (k <
                            // i &&
                            // counterpartColume
                            // > j ) ) {
                            axisLinePoint1 = sortedPointsMetrix[k][l];
                            axisLinePoint2 = sortedPointsMetrix[i][l];
                          }

                          originVector = originPoint
                              .getVectorInReal();

                          if (buildPointInRealByAngle(
                              sortedPointsMetrix[i][j],
                              counterpartPoint,
                              originVector,
                              axisLinePoint1,
                              axisLinePoint2)) {

                            // isRenewed
                            numberOfRenewedPoint++;
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}

boolean buildPointInRealByAngle(Point updatePoint, Point counterpartPoint
     ,
    Vector originVector, Point axisLinePoint1, Point axisLinePoint2) {

  Line axisLine;
  Vector axisVector;
  Vector counterpartVector;
  Vector newVector;
  Vector newVector1;
  Vector newVector2;
  Vector newVector3;

  axisLine = getLine(axisLinePoint1, axisLinePoint2);

  if (axisLine != null
      && (axisLine.getType() == Line.TYPE_POSITIVE_LINE
          || axisLine.getType() == Line.TYPE_NEGATIVE_LINE || axisLine
```

```
            .getType() == Line.TYPE_STATIC_LINE)) {

    } else {
      return false;
    }

    axisVector = new Vector();
    // get axisVector
    // axisVector = axisLine.getVector();
    axisVector.setXYZ(axisLinePoint1.getXInReal()
        - axisLinePoint2.getXInReal(), axisLinePoint1.getYInReal()
        - axisLinePoint2.getYInReal(), axisLinePoint1.getZInReal()
        - axisLinePoint2.getZInReal());

    // get counterpartVector
    counterpartVector = new Vector();
    counterpartVector.setXYZ(counterpartPoint.getXInReal()
        - originVector.getX(), counterpartPoint.getYInReal()
        - originVector.getY(), counterpartPoint.getZInReal()
        - originVector.getZ());

    // get newVector
    newVector1 = new Vector();
    newVector2 = new Vector();
    newVector3 = new Vector();

    newVector1.setXYZ(counterpartVector);
    newVector3.setXYZ(counterpartVector);

    newVector1.rotation(axisVector, (float) ( Math.PI - axisLine.getAngle()
        ));
    newVector3.invert();

    newVector = newVector1;

    if (axisLine.getType() == Line.TYPE_STATIC_LINE) {
      newVector = newVector3;
    }

    newVector.transform(originVector);

    // transform to originVector
    updatePoint.setVectorInReal(newVector);

    // isRenewed
    updatePoint.isRenewed = true;
    return true;
  }

  void transform(Vector vector) {
    transform(vector.getX(), vector.getY(), vector.getZ());
  }

  // transform or rotation whole paper
  void transform(float x, float y, float z) {
    int i;
    for (i = 0; i < numberOfPoints; i++) {
      pointsOnPaper[i].setPointInReal(pointsOnPaper[i].getXInReal() + x,
          pointsOnPaper[i]
          .getYInReal()
          + y, pointsOnPaper[i].getZInReal() + z);
    }
  }

  void rotation(Vector axisVector, float angle) {
    int i;
    Vector tempVector;
    for (i = 0; i < numberOfPoints; i++) {

      // -1 * theta rotation by z axis

      tempVector = pointsOnPaper[i].getVectorInReal();
      tempVector.rotation(axisVector, angle);
      pointsOnPaper[i].setVectorInReal(tempVector);
    }

  }

  void rotation(float r, float theta, float phi, float angle) {

    Vector axisVector = new Vector();

    axisVector.setRThetaPhi(r, theta, phi);

    rotation(axisVector, angle);
  }

  public Line getLine(Point point0, Point point1) {

    int i;

    for (i = 0; i < numberOfLines; i++) {
      if ((point0 == lines[i].getStartPoint() && point1 == lines[i]
          .getEndPoint())
          || (point1 == lines[i].getStartPoint() && point0 == lines[i]
              .getEndPoint())) {

        return lines[i];
      }
    }
    return null;
  }

  // set up sortedPoint
  void buildSortedPoints() {
    int i, j;
    Point tempPoint;

    int numberOfColumes = (int) Math.sqrt((double) numberOfPoints);

    sortedPointsLine = new Point[numberOfPoints];

    for (i = 0; i < numberOfPoints; i++) {
      sortedPointsLine[i] = pointsOnPaper[i];
    }

    for (i = 0; i < numberOfPoints; i++) {
      for (j = i + 1; j < numberOfPoints; j++) {
        if ((sortedPointsLine[i].getYOnPaper() > sortedPointsLine[j]
            .getYOnPaper())
            || ((sortedPointsLine[i].getYOnPaper() == sortedPointsLine[j]
                .getYOnPaper()) && (sortedPointsLine[i]
                .getXOnPaper() >= sortedPointsLine[j]
                .getXOnPaper()))) {
          tempPoint = sortedPointsLine[i];
          sortedPointsLine[i] = sortedPointsLine[j];
          sortedPointsLine[j] = tempPoint;
        }
      }
    }

    sortedPointsMetrix = new Point[numberOfColumes][numberOfColumes];

    for (i = 0; i < numberOfPoints; i++) {
      sortedPointsMetrix[i / numberOfColumes][i % numberOfColumes] =
          sortedPointsLine[i];
    }
  }
```

```java

  public int getNumberOfPoints() {
    return numberOfPoints;
  }

  public int getNumberOfEdges() {
    return numberOfLines;
  }

  public void move(Vector vector) {

  }

  public void rotate(float theta, float pi) {

  }

  // function for Polygons
  void buildPolygons() {
    int i;
    int j;
    int k;

    numberOfPolygons = 0;

    Point[] polygonPoints;

    polygonPoints = new Point[3];

    int numberOfColoums = getNumberOfColoums();

    // build Polygons with clockwise

    for (i = 0; i < numberOfPoints; i++) {
      for (j = i + 1; j < numberOfPoints; j++) {
        if (getLine(pointsOnPaper[i], pointsOnPaper[j]) != null) {
          for (k = j + 1; k < numberOfPoints; k++) {
            if (getLine(pointsOnPaper[j], pointsOnPaper[k]) != null
                && getLine(pointsOnPaper[k], pointsOnPaper[i]) != null) {

              // initiation polygon Points

              // set a polygon with Clockwise

              // if two point's x on paper is same;
              if (pointsOnPaper[j].getXOnPaper() == pointsOnPaper[k]
                  .getXOnPaper()) {

                if (pointsOnPaper[i].getYOnPaper() < pointsOnPaper[j]
                    .getYOnPaper()) {
                  // pick a left point
                  polygonPoints[0] = pointsOnPaper[i];

                  // pick a up point
                  if (pointsOnPaper[j].getYOnPaper() > pointsOnPaper[k]
                      .getYOnPaper()) {

                    polygonPoints[1] = pointsOnPaper[j];
                    polygonPoints[2] = pointsOnPaper[k];

                  } else {

                    polygonPoints[1] = pointsOnPaper[k];
                    polygonPoints[2] = pointsOnPaper[j];

                  }
                } else {

                  // pick a right point
                  polygonPoints[0] = pointsOnPaper[i];

                  // pick the down point
                  if (pointsOnPaper[j].getYOnPaper() < pointsOnPaper[k]
                      .getYOnPaper()) {

                    polygonPoints[1] = pointsOnPaper[j];
                    polygonPoints[2] = pointsOnPaper[k];

                  } else {

                    polygonPoints[1] = pointsOnPaper[k];
                    polygonPoints[2] = pointsOnPaper[j];

                  }
                }
              } else if ((pointsOnPaper[i].getYOnPaper() - pointsOnPaper[j]
                  .getYOnPaper())
                  * (pointsOnPaper[j].getYOnPaper() - pointsOnPaper[k]
                      .getYOnPaper()) < (pointsOnPaper[i]
                  .getXOnPaper() - pointsOnPaper[j].getXOnPaper())
                  * (pointsOnPaper[j].getXOnPaper() - pointsOnPaper[k]
                      .getXOnPaper())) {

                // pick a left down point
                polygonPoints[0] = pointsOnPaper[i];

                if (pointsOnPaper[j].getXOnPaper() > pointsOnPaper[k]
                    .getXOnPaper()) {

                  // pick the up point
                  polygonPoints[1] = pointsOnPaper[j];
                  polygonPoints[2] = pointsOnPaper[k];

                } else {

                  // pick the down point
                  polygonPoints[1] = pointsOnPaper[k];
                  polygonPoints[2] = pointsOnPaper[j];

                }

              } else {
                // pick a right up point
                polygonPoints[0] = pointsOnPaper[i];

                if (pointsOnPaper[j].getYOnPaper() < pointsOnPaper[k]
                    .getYOnPaper()) {

                  // pick a down point
                  polygonPoints[1] = pointsOnPaper[j];
                  polygonPoints[2] = pointsOnPaper[k];

                } else {

                  // pick a up point
                  polygonPoints[1] = pointsOnPaper[k];
                  polygonPoints[2] = pointsOnPaper[j];
                }
              }
              // make polygon
              polygons[numberOfPolygons] = new Polygon();
              try {
                polygons[numberOfPolygons].setPolygon(
                    polygonPoints, this);
              } catch (NoLineException e) {

                e.printStackTrace();
              } catch (Exception e) {

                e.printStackTrace();
              }
              numberOfPolygons++;
            }
          }
        }
      }
    }
  }

  public Polygon getPolygon(int index) {

    if (index < 0 || index > numberOfPolygons) {
      return null;
    }

    return polygons[index];

  }

  public Polygon getPolygon(Point[] point) throws Exception {
    Point polygonPoints[];

    int numberOfMatchPoints;

    int i;
    int j;
    int k;

    for (i = 0; i < numberOfPolygons; i++) {
      polygonPoints = polygons[i].getPoints();
      if (polygonPoints.length == point.length) {
        for (j = 0; j < polygonPoints.length; j++) {
          numberOfMatchPoints = 0;
          for (k = 0; k < polygonPoints.length; k++) {
            if (polygonPoints[k] != point[(k + j)
                % polygonPoints.length]) {
              break;
            }
            numberOfMatchPoints++;
          }
          if (numberOfMatchPoints == polygonPoints.length) {
            return polygons[i];
          }
        }
      }
    }
    return null;
  }

  public Polygon[] getPolygons(Point point) {
    Point[] pointArray = new Point[1];

    pointArray[0] = point;

    return getPolygons(pointArray);
  }

  public Polygon[] getPolygons(Point point0, Point point1) {
    Point[] pointArray = new Point[2];

    pointArray[0] = point0;
    pointArray[1] = point1;

    return getPolygons(pointArray);
  }

  public Polygon[] getPolygons(Point[] points) {
    int i, j, k;
    Polygon[] polygonsHavingPoints;
    Polygon[] oldPolygonsHavingPoints;
    int numberOfPolygonsHavingPoints;

    polygonsHavingPoints = null;
    numberOfPolygonsHavingPoints = 0;
    for (i = 0; i < numberOfPolygons; i++) {
      if (polygons[i].isAllPointsHave(points)) {

        oldPolygonsHavingPoints = polygonsHavingPoints;
        polygonsHavingPoints = new Polygon[numberOfPolygonsHavingPoints +
             1];

        for (j = 0; j < numberOfPolygonsHavingPoints; j++) {
          polygonsHavingPoints[j] = oldPolygonsHavingPoints[j];

        }
        polygonsHavingPoints[numberOfPolygonsHavingPoints] = polygons[i];
        numberOfPolygonsHavingPoints++;

      }
    }
    return polygonsHavingPoints;
  }

  public Polygon getCounterpartPolygon(Polygon polygon, Point point0,
      Point point1) {
    Polygon[] tempPolygons;
    int i;
    Point[] sharedPoints;

    sharedPoints = new Point[2];

    sharedPoints[0] = point0;
    sharedPoints[1] = point1;

    tempPolygons = getPolygons(sharedPoints);

    for (i = 0; i < tempPolygons.length; i++) {
      if (tempPolygons[i] != polygon) {
        return tempPolygons[i];
      }
    }

    return null;
  }

  public Point getCounterpartPointOnCounterpartPolygon(Polygon polygon,
      Point point0, Point point1) {
    Polygon counterpartPolygon;
    int i;
    Point[] pointsOfCounterpartPolygon;

    counterpartPolygon = getCounterpartPolygon(polygon, point0, point1);

    if (counterpartPolygon == null) {
      return null;
    }

    pointsOfCounterpartPolygon = counterpartPolygon.getPoints();

    for (i = 0; i < pointsOfCounterpartPolygon.length; i++) {

      if ((pointsOfCounterpartPolygon[i] != point0)
          && (pointsOfCounterpartPolygon[i] != point1)) {

        return pointsOfCounterpartPolygon[i];
      }
    }

    return null;
  }

  boolean changeAngle(Polygon polygon, Point point0, Point point1,
      float angle) {

    /**
     * 1. if this line is edge line, false 2. get vector from the line
     * changing angle to the oppositePoint from line. 2. rotate vector with
     * angle.
     */

    int i;

    Line line;
    Point changedPoint;
    Polygon counterpartPolygon;

    Vector vector;
    Vector oldVectorInRealOnChangedPoint;
    Vector axisVector;
    Vector originVector;

    Vector va, vb, vc;

    line = getLine(point0, point1);

    if (line.getType() == Line.TYPE_EDGE_LINE) {
      return false;
    }

    if (line.getAngle() == angle) {
      return true;
    }

    // debug monitor

    counterpartPolygon = getCounterpartPolygon(polygon, point0, point1);

    changedPoint = getCounterpartPointOnCounterpartPolygon(polygon, point0,
        point1);

    Vector [] oldVectors = new Vector[getNumberOfPoints()];

    for (i = 0 ; i<getNumberOfPoints(); i++) {
      oldVectors[i] = pointsOnPaper[i].getVectorInReal();
    }

    originVector = counterpartPolygon.getOriginVector(changedPoint, point0,
        point1);

    vector = polygon.getVectorFromLineToOppositePoint(point0, point1);

    // rotation
    axisVector = polygon.getVectorOnTheLine(point0, point1);

    vector.rotation(axisVector.getUnitVector(), ((float) Math.PI) - angle);

    // translate to origin vector;

    vector.addVector(originVector);

    changePointInReal(changedPoint, vector);

    if (fixLinkage(changedPoint)) {

      return true;
    } else {
      for (i = 0 ; i<getNumberOfPoints(); i++) {
        changePointInReal(pointsOnPaper[i], oldVectors[i]);
      }
      return false;
    }
  }

  // adding an angle
  boolean addAngle(Polygon polygon, Point point0, Point point1, float angle
      ) {
    int i;

    Paper paper;
    int polygonIndex;

    Polygon counterpartPolygon;

    Vector unitVectorOnLineOnCounterpatPolygon;
    Vector vectorFromLineToOppositePoint;
    Point oppsitePointOnCounterpatPolygon;

    // error check;
    Line line;
    line = getLine(point0, point1);
    if (line == null) {
      return false;
    }

    if (line.getType() == Line.TYPE_EDGE_LINE || line.getAngle() == 0.0f) {
      return true;
    }

    // get counterpartPolygon
    counterpartPolygon = getCounterpartPolygon(polygon, point0, point1);
    if (counterpartPolygon == null) {
      return true;
    }

    // get unit Vector OnLine
    unitVectorOnLineOnCounterpatPolygon = counterpartPolygon
        .getVectorOnTheLine(point0, point1).getUnitVector();

    // get Vector from line to opposite Point
    vectorFromLineToOppositePoint = counterpartPolygon
        .getVectorFromLineToOppositePoint(point0, point1);

    // rotation the vector
    vectorFromLineToOppositePoint.rotation(
        unitVectorOnLineOnCounterpatPolygon, angle);

    // translate
    vectorFromLineToOppositePoint.transform(polygon
        .getVectorOfStartPointFromLineToOppositePoint(point0, point1));

    // reset the new vector
    oppsitePointOnCounterpatPolygon =
        getCounterpartPointOnCounterpartPolygon(
        polygon, point0, point1);
    oppsitePointOnCounterpatPolygon
        .setVectorInReal(vectorFromLineToOppositePoint);

    return fixLinkage(oppsitePointOnCounterpatPolygon);
  }

  public boolean changePointInReal(Point point, Vector vector) {

    point.setPointInReal(vector.getX(), vector.getY(), vector.getZ());

    // 1 0 0 -1 0 0
    // check the angles

    // is type of angle is different from line type.

    return true;

  }

  public void resetAngleOnLineByPoints(Line line) {

  }

  boolean checkError(Point changedPoint) {
    if (!checkPointError(changedPoint)) {
      return false;
    }
    if (!checkAngleError(changedPoint)) {
      return false;
    }

    return true;
  }

  boolean checkAngleError(Point changedPoint) {
    return true;
  }

  boolean checkPointError(Point changedPoint) {
    /*
     * error check
     *
     * find polygon shared the changed point if there are two line change
         in
     * one polygon return false.
     *
     * if not fix the linkage set point 0 = changedPoint set point 1 until
     * there are no error -> return true;
     *
     * or return false and role back;
     */

    int i;
    int j;
    int k;

    float lengthOfFirstLineOnPaper;
    float lengthOfFirstLineInReal;

    float lengthOfSecondLineOnPaper;
    float lengthOfSecondLineInReal;

    Polygon[] polygonsHavingThisPoint;

    Point[] pointsOfPolygon;
    Point errorPoint;
    Polygon errorPolygon;
    Point pointOnLineInErrorPolygon0;
    Point pointOnLineInErrorPolygon1;
    Point counterpartPointOnCounterpartPolygon;
    Point pointOfOrigin;

    Point pointOnFirstLine;
    Point pointOnSecondLine;

    Vector[] possiableVector;

    // get polygons having error point
    polygonsHavingThisPoint = getPolygons(changedPoint);

    if(polygonsHavingThisPoint.length == 0) {
      int t=0;
    }

    // check the polygons are fixable or not.
    for (i = 0; i < polygonsHavingThisPoint.length; i++) {
      pointsOfPolygon = polygonsHavingThisPoint[i].getPoints();
      errorPolygon = polygonsHavingThisPoint[i];

      for (j = 0; j < Polygon.DEFAULT_NUMBER_OF_POINTS; j++) {
        if (pointsOfPolygon[j] == changedPoint) {

          // get length of first line
          pointOnFirstLine = pointsOfPolygon[(j + 1) % 3];

          lengthOfFirstLineOnPaper = getLine(pointsOfPolygon[j],
               pointOnFirstLine)
              .getLengthOnPaper();

          lengthOfFirstLineInReal = getLine(pointsOfPolygon[j],
              pointOnFirstLine).getLengthInReal();

          // get length of second line
          pointOnSecondLine = pointsOfPolygon[(j + 2) % 3];

          lengthOfSecondLineOnPaper = getLine(pointsOfPolygon[j],
              pointOnSecondLine).getLengthOnPaper();

          lengthOfSecondLineInReal = getLine(pointsOfPolygon[j],
              pointOnSecondLine).getLengthInReal();

          if (lengthOfFirstLineInReal == Float.NaN
              ||  lengthOfSecondLineInReal == Float.NaN ) {
            int t=0;

          }


          // two line error check
          if (!(Math.abs(lengthOfFirstLineInReal
              - lengthOfFirstLineOnPaper) <=
               ERROR_RATIO_FOR_LANGTH)
            && !(Math.abs(lengthOfSecondLineInReal
              - lengthOfSecondLineOnPaper) <=
                ERROR_RATIO_FOR_LANGTH)) {
            return false;
          }

          if (lengthOfFirstLineInReal > 0.36
              || lengthOfSecondLineInReal > 0.36
             || lengthOfFirstLineInReal < 0
              || lengthOfSecondLineInReal < 0){
                int q=0;
          }
        }
      }

    }


    return true;
  }

  boolean fixLinkage(Point changedPoint) {
    /*
     * error check
     *
```

```
   * find polygon shared the changed point if there are two line change
        in
   * one polygon return false.
   *
   * if not fix the linkage set point 0 = changedPoint set point 1 until
   * there are no error -> return true;
   *
   * or return false and role back;
   */

  int i;
  int j;
  int k;

  float lengthOfFirstLineOnPaper;
  float lengthOfFirstLineInReal;

  float lengthOfSecondLineOnPaper;
  float lengthOfSecondLineInReal;

  Polygon[] polygonsHavingThisPoint;

  Point[] pointsOfPolygon;
  Point errorPoint;
  Polygon errorPolygon;
  Point pointOnLineInErrorPolygon0;
  Point pointOnLineInErrorPolygon1;
  Point counterpartPointOnCounterpartPolygon;
  Point pointOfOrigin;

  Point pointOnFirstLine;
  Point pointOnSecondLine;

  Vector[] possibleVector;

  if (!checkError(changedPoint)) {
    return false;
  }

  // get polygons having error point
  polygonsHavingThisPoint = getPolygons(changedPoint);

  if (polygonsHavingThisPoint.length < 1) {
    int t =0;
  }

  for (i = 0; i < polygonsHavingThisPoint.length; i++) {
    pointsOfPolygon = polygonsHavingThisPoint[i].getPoints();
    errorPolygon = polygonsHavingThisPoint[i];
    // two line error
    for (j = 0; j < Polygon.DEFAULT_NUMBER_OF_POINTS; j++) {
      if (pointsOfPolygon[j] == changedPoint) {

        // get length of first line
        pointOnFirstLine = pointsOfPolygon[(j + 1) % 3];

        lengthOfFirstLineOnPaper = getLine(pointsOfPolygon[j],
            pointOnFirstLine)
            .getLengthOnPaper();

        lengthOfFirstLineInReal = getLine(pointsOfPolygon[j],
            pointOnFirstLine)
            .getLengthInReal();

        // get length of second line
        pointOnSecondLine = pointsOfPolygon[(j + 2) % 3];

        lengthOfSecondLineOnPaper = getLine(pointsOfPolygon[j],
            pointOnSecondLine).getLengthOnPaper();

        lengthOfSecondLineInReal = getLine(pointsOfPolygon[j],
            pointOnSecondLine).getLengthInReal();

        if (!(Math.abs(lengthOfFirstLineInReal
            - lengthOfFirstLineOnPaper) <=
             ERROR_RATIO_FOR_LANGTH)
            || !(Math.abs(lengthOfSecondLineInReal
                - lengthOfSecondLineOnPaper) <=
                  ERROR_RATIO_FOR_LANGTH)) {

          // find error point
          if (!(Math.abs(lengthOfFirstLineInReal
              - lengthOfFirstLineOnPaper) <=
               ERROR_RATIO_FOR_LANGTH)) {
            errorPoint = pointOnFirstLine;
            changedPoint = changedPoint;
            pointOfOrigin = pointOnSecondLine;

          } else {
            errorPoint = pointOnSecondLine;
            changedPoint = changedPoint;
            pointOfOrigin = pointOnFirstLine;
          }

          counterpartPointOnCounterpartPolygon =
               getCounterpartPointOnCounterpartPolygon(
              errorPolygon, errorPoint, pointOfOrigin);

          if (errorPoint == null
              || changedPoint == null
              || pointOfOrigin == null
              || counterpartPointOnCounterpartPolygon== null  ){
            return false;
          }

          possibleVector = getVectorsOfPointsOnThreePoints(
              errorPoint, changedPoint, pointOfOrigin,
              counterpartPointOnCounterpartPolygon);

          float[] energy = new float[2];

          changePointInReal(errorPoint, possibleVector[0]);

          if (!fixLinkage(errorPoint)) {
            changePointInReal(errorPoint, possibleVector[1]);
            if (!fixLinkage(errorPoint)) {
              return false;
            }
          } else {
            energy[0] = getGlobalEnergy();
            changePointInReal(errorPoint, possibleVector[1]);
            if (fixLinkage(errorPoint)) {
              energy[1] = getGlobalEnergy();
              if (energy[0] < energy[1]) {
                changePointInReal(errorPoint,
                    possibleVector[0]);
              }
            } else {
              changePointInReal(errorPoint, possibleVector[0]);
            }
          }
        }
      }
      if(j>=3) {
        int o=0;
      }
    }
  }
```

```
    // error 2:
    // inside line -> outside line
    // outside line -> inside line
    // 0-> inside
    // 0-> outside
/**/
    for(i = 0; i<getNumberOfEdges(); i++ ){
      if (!(Math.abs(lines[i].getLengthInReal()- lines[i].getLengthOnPaper
          ()) < ERROR_RATIO_FOR_LANGTH)){
/** /
        System.out.format("line length difference is biger than %f
             LengthInReal = %f LentghOnPaper = %f difference = %f\n"
            , ERROR_RATIO_FOR_LANGTH
            ,lines[i].getLengthInReal()
            ,lines[i].getLengthOnPaper()
            ,lines[i].getLengthInReal() - lines[i].getLengthOnPaper());
/**/
        return false;
      }
      if (!(lines[i].getLengthInReal()< 3.7 )
          || !(lines[i].getLengthOnPaper() < 3.7 )) {
/** /
        System.out.format("line length is biger than 3.7 LengthInReal = %f
             LentghOnPaper = %f\n"
            ,lines[i].getLengthInReal()
            ,lines[i].getLengthOnPaper());
/**/
        return false;
      }
    }
/**/
    return true;
  }

  boolean fixAngleOnPolygons(Polygon polygon, Line line) {
    /*
     * return false : Way of angle is changed .; return true : way of Angle
     * not changed ;
     */
    Polygon[] polygonsSharedLine;
    Point point0;
    Point point1;

    Vector vectorOfLineOnPolygon;
    Vector vectorFromClossProduct;

    float oldAngle;
    float newAngle;

    Vector unitVectorPolygon;
    Vector unitVectorCounterpartPolygon;

    point0 = line.getStartPoint();
    point1 = line.getEndPoint();

    unitVectorPolygon = polygon.getVectorFromLineToOppositePoint(point0,
        point1).getUnitVector();
    unitVectorCounterpartPolygon = getCounterpartPolygon(polygon, point0,
        point1).getVectorFromLineToOppositePoint(point0, point1)
        .getUnitVector();

    unitVectorPolygon.invert();
    unitVectorPolygon = unitVectorPolygon.getUnitVector();

    oldAngle = line.getAngle();
    newAngle = (float) Math.acos(Vector.dot(unitVectorPolygon,
        unitVectorCounterpartPolygon));

    vectorOfLineOnPolygon = polygon.getVectorOnTheLine(point0, point1);
```

```
    vectorFromClossProduct = Vector.closs(unitVectorPolygon,
        unitVectorCounterpartPolygon);

    // same way to line vector
    if (Vector.dot(unitVectorPolygon, unitVectorPolygon) > 0.99999999) {

    } else {
      newAngle = -1 * newAngle;
    }

    if (oldAngle * newAngle < 0) {

      return false;
    } else {

      return true;
    }
  }

  // function for snapshot
  public Paper snapshot() {
    int i;
    Paper paper = new Paper();

    Point[] points = new Point[numberOfPoints];
    for (i = 0; i < this.numberOfPoints; i++) {
      points[i] = this.pointsOnPaper[i].snapshot();
    }

    Line[] lines = new Line[numberOfLines];
    for (i = 0; i < this.numberOfLines; i++) {
      lines[i] = this.lines[i].snapshot(paper, points);
    }

    Polygon[] polygons = new Polygon[numberOfPolygons];
    for (i = 0; i < this.numberOfPolygons; i++) {
      polygons[i] = this.polygons[i].snapshot(paper, points);
    }

    paper.setValue(points, numberOfPoints, lines, numberOfLines, polygons,
        numberOfPolygons);

    return paper;
  }

  public boolean changeAngleAsMuchAsPossible(Polygon polygon, Point point0,
      Point point1) {
    /**
     * Unfolding Edge as much as possible
     *
     */

    int i;

    boolean isAngleChanged;

    float targetAngle;
    float stepAngle;

    Line line;
    float foldingWay;

    line = getLine(point0, point1);

    if (line.getType() == Line.TYPE_POSITIVE_LINE) {
      foldingWay = 1.0f;
    } else if (line.getType() == Line.TYPE_NEGATIVE_LINE) {
      foldingWay = -1.0f;
    } else {
      return false;
    }

    if (changeAngle(polygon, point0, point1, (float) 0)) {
      return true;
    }

    stepAngle = (float) Math.PI / 2;
    targetAngle = (float) Math.PI / 2;

    isAngleChanged = false;
    while (stepAngle != 0.0f) {

      stepAngle /= 2;

      if (changeAngle(polygon, point0, point1, foldingWay * targetAngle)) {

        isAngleChanged = true;
        targetAngle -= stepAngle;

      } else {

        targetAngle += stepAngle;
      }
    }

    return isAngleChanged;
  }

  public boolean changeAngleForSmallestGlobalEnergy(Line line){
    boolean isAnglechanged = true;

    return isAnglechanged;
  }

  /**
   *
   * Unfolding Edge by target angles.
   *
   * 1. copy the paper. 2. pick a line. set the angle on line the target
   * point. 3. fix The Other Point around the point. 4. if fixing is false,
   * roll back. false state is there is more than 1 edge in one polygon we
   * have to fix. if fixing is success repeat 2 until there is no angle.
   *
   * Unfolding Edge by step
   */

  public void setValue(Point[] points, int numberOfPoints, Line[] lines,
      int numberOfLines, Polygon[] polygons, int numberOfPolygons) {

    this.pointsOnPaper = points;
    this.numberOfPoints = numberOfPoints;
    this.lines = lines;
    this.numberOfLines = numberOfLines;
    this.polygons = polygons;
    this.numberOfPolygons = numberOfPolygons;

    buildSortedPoints();

  }

  public void setValue(Paper paper) {

    setValue(paper.getPoints(), paper.getNumberOfPoints(),
        paper.getLines(), paper.getNumberOfEdges(),
        paper.getPolygons(), paper.getNumberOfPolygons());

  }

  Vector[] getVectorsOfPointsOnThreePoints(Point errorPoint, Point point0,
      Point point1, Point point2) {
    Vector v1, v2, v3;
    float length1, length2, length3;

    v1 = point0.getVectorInReal();
    v2 = point1.getVectorInReal();
    v3 = point2.getVectorInReal();

    length1 = getLine(point0, errorPoint).getLengthOnPaper();
    length2 = getLine(point1, errorPoint).getLengthOnPaper();
    length3 = getLine(point2, errorPoint).getLengthOnPaper();

    return Vector.getVectorFrom3Vector(v1, length1, v2, length2, v3,
        length3);
  }

  public float getGlobalEnergy() {

    // energy = (float) Math.pow(energy,2) - 1.0f ;

    // System.out.printf("Energy = %f\n " , energy);
    return getEnergy(pointsOnPaper);
  }

  public void printAllOfLineLength() {
    int i;
    for (i = 0; i < numberOfLines; i++) {
      System.out.format("l in real =%f l on paper=%f \n", lines[i]
          .getLengthInReal(), lines[i].getLengthOnPaper());
    }
    System.out.format("\n");
  }

  public float getLocalEnergy(Line line) {

    int i, j;

    ArrayList<Point> pointArray = new ArrayList<Point>();


    Point startPoint;
    Point endPoint;

    pointArray.add(line.getStartPoint());
    pointArray.add(line.getEndPoint());

    boolean isStartPointInPointArray = false;
    boolean isEndPointInPointArray = false;


    for (i=0; i <getNumberOfEdges(); i++) {

      isStartPointInPointArray = false;
      isEndPointInPointArray = false;

      startPoint = getLine(i).getStartPoint();
      endPoint = getLine(i).getEndPoint();
      for (j=0; j<pointArray.size(); j++) {
        if(startPoint == pointArray.get(j)) {
          isStartPointInPointArray = true;
        }
        if(endPoint == pointArray.get(j)) {
          isEndPointInPointArray = true;
        }
      }

      if (!(isStartPointInPointArray & isEndPointInPointArray)
          && !isStartPointInPointArray){
```

```
        pointArray.add(startPoint);
      } else if (!(isStartPointInPointArray & isEndPointInPointArray)
         && !isEndPointInPointArray) {
        pointArray.add(endPoint);
      }
    }

    return getEnergy((Point[]) pointArray.toArray());
  }

  public static float getEnergy(Point[] points) {
    int i;
    int j;

    float energy = 0.0f;

    for (i = 0; i < points.length; i++) {
      for (j = 0; j < points.length; j++) {
        energy += (points[i].getXInReal() - points[j].getXInReal())
            * (points[i].getXInReal() - points[j].getXInReal())
            * (points[i].getXInReal() - points[j].getXInReal())
            * (points[i].getXInReal() - points[j].getXInReal())
            + (points[i].getYInReal() - points[j].getYInReal())
            * (points[i].getYInReal() - points[j].getYInReal())
            * (points[i].getYInReal() - points[j].getYInReal())
            * (points[i].getYInReal() - points[j].getYInReal())
            + (points[i].getZInReal() - points[j].getZInReal())
            * (points[i].getZInReal() - points[j].getZInReal())
            * (points[i].getZInReal() - points[j].getZInReal())
            * (points[i].getZInReal() - points[j].getZInReal());

        energy -= (points[i].getXOnPaper() - points[j].getXOnPaper())
            * (points[i].getXOnPaper() - points[j].getXOnPaper())
            * (points[i].getXOnPaper() - points[j].getXOnPaper())
            * (points[i].getXOnPaper() - points[j].getXOnPaper())
            + (points[i].getYOnPaper() - points[j].getYOnPaper())
            * (points[i].getYOnPaper() - points[j].getYOnPaper())
            * (points[i].getYOnPaper() - points[j].getYOnPaper())
            * (points[i].getYOnPaper() - points[j].getYOnPaper())
            + (points[i].getZOnPaper() - points[j].getZOnPaper())
            * (points[i].getZOnPaper() - points[j].getZOnPaper())
            * (points[i].getZOnPaper() - points[j].getZOnPaper())
            * (points[i].getZOnPaper() - points[j].getZOnPaper());
      }
    }

    energy = energy * energy;

    return energy;
  }
}
```

```
package com.drancom.programmableMatter.folding.controller.paper;

import com.drancom.programmableMatter.folding.controller.paper.util.Vector;

public class Polygon {
  public final static int DEFAULT_NUMBER_OF_POINTS = 3;

  Paper paper;

  Point points[];
  int numberOfPoints;


  public void setPolygon (Point[] points, Paper paper) throws Exception,
        NoLineException  {

    int i;

    int numberOfPoints;
    int index_smallestCoordinatePoint;

    Vector vector_smallestCoordinatePoint;
    Vector vector;

    this.paper = paper;

    numberOfPoints = points.length;

    for (i = 0; i < numberOfPoints; i++) {
      if(paper.getLine(points[i], points [(i + 1) % numberOfPoints])== null
          ){
        throw new NoLineException();
      }
    }

    // find point having on the smallest  z < y < x ;
    index_smallestCoordinatePoint = 0;
    vector_smallestCoordinatePoint = points[index_smallestCoordinatePoint].
          getVectorOnPaper();

    for (i = 0; i < numberOfPoints; i++) {
      vector = points[i].getVectorOnPaper();
      if(((vector_smallestCoordinatePoint.getZ() < vector.getZ())
          || (vector_smallestCoordinatePoint.getZ() == vector.getZ() )
             &&   vector_smallestCoordinatePoint.getY() < vector.getY())
          || (vector_smallestCoordinatePoint.getZ() == vector.getZ()
             &&   vector_smallestCoordinatePoint.getY() == vector.getY()
             &&   vector_smallestCoordinatePoint.getX() < vector.getX()))
                  {
        index_smallestCoordinatePoint=i;
        vector_smallestCoordinatePoint = points[
             index_smallestCoordinatePoint].getVectorOnPaper();
      }
    }

    this.points = new Point[numberOfPoints];

    for(i = 0; i < numberOfPoints; i++) {
      this.points[i] = points[(i+index_smallestCoordinatePoint) %
            numberOfPoints];
      this.numberOfPoints = numberOfPoints;
    }
  }

  public Point[] getPoints(){
    return points;
  }

  public int getNumberOfPoints() {
    return numberOfPoints;
  }

  public Point getOppositePoint(Point point0, Point point1) {
    int i;
    for(i=0; i<3; i++) {
      if(points[i] != point0 && points[i] != point1)
        return points[i];
    }
    return null;
  }

  public Vector getVectorOnTheLine(Point point0, Point point1) {
    int i;
    Vector vectorOnTheLine;
    for(i=0 ; i < numberOfPoints; i++) {
      if((points[i] == point0 && points[(i+1) % numberOfPoints] == point1)
        || (points[i] == point1 && points[(i+1) % numberOfPoints] == point0
            )){
        vectorOnTheLine = new Vector();
        vectorOnTheLine.setXYZ(points[(i+1) % numberOfPoints].
            getVectorInReal());

        vectorOnTheLine.subtractionVector(points[i].getVectorInReal());
        return vectorOnTheLine;
      }
    }
    return null;
  }

  public Vector getVectorFromLineToOppositePoint (Point point0, Point
      point1) {
    int i;
    Point oppositPoint;
    Vector vectorFromLineToOppsitePoint;
    Vector originVector;
    if(null == paper.getLine(point0, point1)){
      return null;
    }

    // vectorOnLineForStartPointOfVectorFromLineToOppsitePoint
    oppositPoint = getOppositePoint(point0, point1);

    originVector = getOriginVector(oppositPoint,
        point0,
        point1);


    vectorFromLineToOppsitePoint = oppositPoint.getVectorInReal();

    originVector.invert();
    vectorFromLineToOppsitePoint.addVector(originVector);

    return vectorFromLineToOppsitePoint;
  }
  public Vector getVectorOfStartPointFromLineToOppositePoint(Point point0,
      Point point1) {
    Vector vectorFromStartPointToOppsitePoint;
    Vector vectorOnLine;

    Vector unitVectorAngle;

    Vector vectorOnLineForStartPointOfVectorFromLineToOppsitePoint;


    float angle;
    float scaleOfVectorOnLineForStartPointOfVectorFromLineToOppsitePoint;


    vectorOnLine = new Vector();

    vectorFromStartPointToOppsitePoint = new Vector();


    // end Point
    vectorFromStartPointToOppsitePoint.setXYZ(point0.getVectorInReal());
    // start point
    vectorFromStartPointToOppsitePoint.subtractionVector(getOppositePoint(
        point0, point1).getVectorInReal());

    // end Point
    vectorOnLine.setXYZ(point0.getVectorInReal());
    // start Point
    vectorOnLine.subtractionVector(getOppositePoint(point0, point1).
        getVectorInReal());

    // find angle
    angle = (float) Math.acos((double) Vector.dot(
        vectorFromStartPointToOppsitePoint.getUnitVector(), vectorOnLine.
        getUnitVector()));

    // find startPoint for Vector from line to opposite Point
    scaleOfVectorOnLineForStartPointOfVectorFromLineToOppsitePoint = (
        float) ((float) vectorFromStartPointToOppsitePoint.getR() * Math.
        cos((float) angle));
    vectorOnLineForStartPointOfVectorFromLineToOppsitePoint = vectorOnLine.
        getUnitVector();
    vectorOnLineForStartPointOfVectorFromLineToOppsitePoint.scale(
        scaleOfVectorOnLineForStartPointOfVectorFromLineToOppsitePoint);

    return vectorOnLineForStartPointOfVectorFromLineToOppsitePoint;
  }
  public boolean isAllPointsHave(Point point0, Point point1, Point point2)
      {
    Point[] pointsAll = new Point[3];

    pointsAll[0]  = point0;
    pointsAll[1]  = point1;
    pointsAll[2]  = point2;


    return isAllPointsHave(pointsAll);
  }

  public boolean isAllPointsHave(Point[] points) {
    int i;
    int j;

    int numberOfmachingPoints;
    numberOfmachingPoints = 0;

    for (i=0; i < points.length ; i++) {
      for (j=0; j < this.points.length ; j++) {
        if(this.points[j] == points[i]) {
          numberOfmachingPoints++;
          if(numberOfmachingPoints == points.length){
            return true;
          }
        }
      }
    }
    return false;
  }

  public boolean setAngle(Point point0, Point point1, float angle) throws
      NoLineException{
    int i;
    Line line;

    for(i=0; i < numberOfPoints; i++) {
      if ((points[i] == point0 && points[ ( i + 1 ) % numberOfPoints] ==
           point1)
          || (points[i] == point1 && points[ ( i + 1) % numberOfPoints] ==
              point0)) {

        return paper.changeAngle(this, point0, point1, angle);
      }
    }

    throw new NoLineException();

  }
  public boolean isPointHaving(Point point) {
    int i;

    for(i=0; i < numberOfPoints; i++) {
      if (points[i] == point){
        return true;
      }
    }
    return false;
  }

  public Vector getOriginVector(Point point0, Point point1) {
    Point oppositePoint;

    oppositePoint = getOppositePoint(point0, point1);

    return getOriginVector(oppositePoint, point0, point1);

  }
  public Vector getOriginVector(Point oppositePoint, Point point0, Point
        point1) {
    if (!isAllPointsHave(oppositePoint, point0, point1)){
      return null;
    }

    Vector originVector;

    int i;
    Vector va, vb, vc;
    float dot;

    //Vo = V0 + (V1-V0)unit * lengthOf(Vold - V0) * (Vold - V0).unit dot (
          V1 - V0).unit
    //Va = Vold-V0;
    //Vb = V1-V0;
    //Vc = Vb unit * lengthOf(Va) * cos(acos((Va).unit dot (Vb).unit )))
    //Vo = V0 + Vc;
    va = new Vector();
    vb = new Vector();
    vc = new Vector();

    va.setXYZ(point0.getVectorInReal());
    vb.setXYZ(point0.getVectorInReal());

    va.invert();
    vb.invert();

    va.addVector(oppositePoint.getVectorInReal());
    vb.addVector(point1.getVectorInReal());

    vc.setXYZ(vb.getUnitVector());
    dot = (Vector.dot(va.getUnitVector(), vb.getUnitVector()));
    vc.scale( (float) (va.getR() * Math.cos(Vector.acos(dot))));

    originVector = point0.getVectorInReal();
    originVector.addVector(vc);


    return originVector;
  }


  public boolean resetAngle(Point point0, Point point1) {



    return false;
  }
  public Polygon snapshot(Paper newPaper, Point[] newPoints) {
    int i;
    int j;

    Polygon polygon = new Polygon();
    Point[] newPointsInPolygon = new Point[getNumberOfPoints()];

    Point[] newSetOfPoints = newPoints;

    for ( i = 0 ; i < getNumberOfPoints() ; i++) {
      for(j=0; j < paper.getNumberOfPoints(); j++){
        if(this.points[i].getXOnPaper() == newSetOfPoints[j].getXOnPaper()
            && this.points[i].getYOnPaper() == newSetOfPoints[j].
                getYOnPaper()
            && this.points[i].getZOnPaper() == newSetOfPoints[j].
                getZOnPaper()){
          newPointsInPolygon[i] = newSetOfPoints[j];
        }
      }
    }

    polygon.setValues(newPaper, newPointsInPolygon, numberOfPoints);
    return polygon;
  }

  void setValues(
      Paper paper
      ,Point points[]
      , int numberOfPoints) {


    this.paper = paper;

    this.points = points;
    this.numberOfPoints = numberOfPoints;


  }

}
```

```
package com.drancom.programmableMatter.folding.controller.paper;

import com.drancom.programmableMatter.folding.controller.paper.util.Vector;

public class Line {

  Paper paper;
  public final static int DIRECTION_STRIGHT_LINE = 0;
  public final static int DIRECTION_OBLIQUE_LINE = 1;

  public final static int TYPE_STATIC_LINE = 0;
  public final static int TYPE_EDGE_LINE = 1;
  public final static int TYPE_POSITIVE_LINE = 2;
  public final static int TYPE_NEGATIVE_LINE = 3;
  public final static int TYPE_BOTHWAY_LINE = 4;


  public final static int MAX_POSITIVE_ANGLE = 180;
  public final static int MAX_NEGATIVE_ANGLE = 180;


  float maxPositiveAngle = 0;
  float maxNegativeAngle = 0;

  Point startPoint;
  Point endPoint;

  float angle;

  int type;
  int directionOfLine;

  float levelOfActuratingPower = 0.0f;

  Vector getVector() {
    Vector vector = new Vector();
    vector.setXYZ(endPoint.getXInReal() - startPoint.getXInReal()
          , endPoint.getYInReal() - startPoint.getYInReal()
          , endPoint.getZInReal() - startPoint.getZInReal());

    return vector;
  }
  public int getType(){
    return type;
  }

  public Point getStartPoint(){
    return startPoint;
  }

  public Point getEndPoint(){
    return endPoint;
  }

  public float getAngle(){
    int i;
/** /
    float angleBetweenVectors;
    float angle;
    Polygon[] polygons;
    Vector v1, v2;

    if (type == TYPE_EDGE_LINE) {
      return Float.intBitsToFloat(0x7fc00000); //Not a number
    }

    polygons = paper.getPolygons(startPoint, endPoint);
    if (polygons.length != 2) {
      return Float.intBitsToFloat(0x7fc00000); //Not a number
    }

    v1= polygons[0].getVectorFromLineToOppositePoint(startPoint, endPoint);
    v2= polygons[1].getVectorFromLineToOppositePoint(startPoint, endPoint);
    angleBetweenVectors = (float) Math.acos(Vector.dot(v1, v2));

    if (type == TYPE_POSITIVE_LINE){
      angle = (float) Math.PI - angleBetweenVectors;
    } else  if (type == TYPE_NEGATIVE_LINE){
      angle = -1 * ( (float) Math.PI -  angleBetweenVectors);
    } else {
      angle = (float) Math.PI - angleBetweenVectors;
    }
/**/
//    this.angle = angle;

    return angle;

  }

  public float getLengthOnPaper() {
    Vector v1, v2;
    v1 = startPoint.getVectorOnPaper();
    v2 = endPoint.getVectorOnPaper();
    v2.invert();
    v1.addVector(v2);

    return  v1.getR();
  }

  public float getLengthInReal() {
    Vector v1, v2;
    v1 = startPoint.getVectorInReal();
    v2 = endPoint.getVectorInReal();
    v2.invert();
    v1.addVector(v2);

    return  v1.getR();
  }

  public void setLevelOfActuratingPower(float levelOfActuratingPower){
    this.levelOfActuratingPower = levelOfActuratingPower;
  }

  public float getLevelOfActuratingPower(){
    return levelOfActuratingPower;
  }

  public boolean isActurating(){

    if (levelOfActuratingPower==0.0) {

      return false;

    } else {
      return true;
    }
  }


  public void setAngle(float angle) {

    this.angle = angle;

    if(this.angle == 0.0) {
      this.type = TYPE_STATIC_LINE;
    } else if(angle > 0.0) {
      this.type = TYPE_POSITIVE_LINE;
    } else {
      this.type = TYPE_NEGATIVE_LINE;
    }
  }

  public void addAngle(float angle) {

    this.angle += angle;

    if(this.angle == 0.0) {
      this.type = TYPE_STATIC_LINE;
    } else if(angle > 0.0) {
      this.type = TYPE_POSITIVE_LINE;
    } else {
      this.type = TYPE_NEGATIVE_LINE;
    }
  }

  public void setLine(Paper paper,
            Point startPoint,
            Point endPoint,
            float angle,
            int typeOfLine){

    this.paper = paper;


    if (startPoint.getXOnPaper() == endPoint.getXOnPaper()) {
      if (startPoint.getYOnPaper() < endPoint.getYOnPaper()) {

        this.startPoint = startPoint;
        this.endPoint = endPoint;

      } else {

        this.startPoint = endPoint;
        this.endPoint = startPoint;

      }
    } else if (startPoint.getYOnPaper() == endPoint.getYOnPaper()) {
      if (startPoint.getXOnPaper() < endPoint.getXOnPaper()) {

        this.startPoint = startPoint;
        this.endPoint = endPoint;

      } else {
        this.startPoint = endPoint;
        this.endPoint = startPoint;
      }

    } else if (startPoint.getXOnPaper() < endPoint.getXOnPaper()) {
      this.startPoint = startPoint;
      this.endPoint   = endPoint;
    } else {
      this.startPoint = endPoint;
      this.endPoint   = startPoint;
    }


    this.angle = angle;
    this.type = typeOfLine;

    if(startPoint.getXOnPaper() == endPoint.getXOnPaper()
         || startPoint.getYOnPaper() == endPoint.getYOnPaper()){

      this.directionOfLine = DIRECTION_STRIGHT_LINE;

    } else{
      this.directionOfLine = DIRECTION_OBLIQUE_LINE;
      startPoint.setType(Point.TYPE_OBLIQUE_LINE_CLOSS);
      endPoint.setType(Point.TYPE_OBLIQUE_LINE_CLOSS);
    }



    if (typeOfLine == TYPE_STATIC_LINE){
      maxPositiveAngle = 0;
      maxNegativeAngle = 0;
    }

    if (typeOfLine == TYPE_POSITIVE_LINE){
      maxPositiveAngle = 180;
      maxNegativeAngle = 0;
    }

    if (typeOfLine == TYPE_NEGATIVE_LINE){
      maxPositiveAngle = 0;
      maxNegativeAngle = 180;
    }

    if (typeOfLine == TYPE_BOTHWAY_LINE){
      maxPositiveAngle = 180;
      maxNegativeAngle = 180;
    }
  }

  public Line snapshot(Paper paper, Point [] newSetOfPoints) {
    int i;

    Line line = new Line();
    Point startPoint = null;
    Point endPoint = null;


    for(i=0; i<newSetOfPoints.length; i++){
      if(newSetOfPoints[i].getXOnPaper() == this.startPoint.getXOnPaper()
          && newSetOfPoints[i].getYOnPaper() == this.startPoint.getYOnPaper
               ()
          && newSetOfPoints[i].getZOnPaper() == this.startPoint.getZOnPaper
               ()){
        startPoint = newSetOfPoints[i];
      } else if(newSetOfPoints[i].getXOnPaper() == this.endPoint.
            getXOnPaper()
            && newSetOfPoints[i].getYOnPaper() == this.endPoint.getYOnPaper
                 ()
            && newSetOfPoints[i].getZOnPaper() == this.endPoint.getZOnPaper
                 ()){
          endPoint = newSetOfPoints[i];
      }

      if(startPoint != null && endPoint != null ){
        line.setValues(paper, maxPositiveAngle, maxNegativeAngle,
             startPoint, endPoint, angle, type, directionOfLine);
        return line;
      }
    }

    return null;
  }

  void setValues(
      Paper paper
      ,float maxPositiveAngle
      ,float maxNegativeAngle

      , Point startPoint
      , Point endPoint

      ,float angle

      , int type
      , int directionOfLine) {

    this.paper = paper;

    this.maxPositiveAngle = maxPositiveAngle;
    this.maxNegativeAngle = maxNegativeAngle;

    this.startPoint = startPoint;
    this.endPoint   = endPoint;

    this.angle = angle;

    this.type = type;
    this.directionOfLine = directionOfLine;

  }
}
```

```
package com.drancom.programmableMatter.folding.controller.paper;

import com.drancom.programmableMatter.folding.controller.paper.util.Vector;

public class Point {

  public final static int PLACE_OF_DECIMAL_POINT = 4;

  public final static int TYPE_ONLY_STRIGHT_LINE_CLOSS  = 0;
  public final static int TYPE_OBLIQUE_LINE_CLOSS       = 1;

  boolean isRenewed;
```

```java

  Vector vOnPaper;

  Vector vInReal;


  private int type;

  public Point () {
    isRenewed = false;
    vOnPaper = new Vector();
    vInReal  = new Vector();

    type = TYPE_ONLY_STRIGHT_LINE_CLOSS;
  }

  public float getXOnPaper() {
    return vOnPaper.getX();
  }

  public float getYOnPaper() {
    return vOnPaper.getY();
  }

  public float getZOnPaper() {
    return vOnPaper.getZ();
  }

  public float getXInReal() {
    return vInReal.getX();
  }

  public float getYInReal() {
    return vInReal.getY();
  }

  public float getZInReal() {
    return vInReal.getZ();
  }

  public void setType(int type){
    this.type = type;
  }

  public int getType(){
    return type;
  }


  public Vector getVectorInReal (){
    Vector vector = new Vector();

    vector.setXYZ(vInReal);

    return vector;
  }

  public Vector getVectorOnPaper (){
    Vector vector = new Vector();

    vector.setXYZ(vOnPaper);

    return vector;
  }

  public void setRenewed(boolean isRenewed) {
    this.isRenewed = isRenewed;
  }


  public boolean isRenewed(){
    return isRenewed;
  }

  public void setPointOnPaper(float xOnPaper, float yOnPaper, float
       zOnPaper) {
    vOnPaper.setXYZ(xOnPaper,yOnPaper,zOnPaper);
  }

  public void setPointInReal (float xInReal,  float yInReal,  float zInReal
       ) {
    vInReal.setXYZ(xInReal, yInReal, zInReal);
  }

  public void setVectorInReal (Vector vector){
    setPointInReal(vector.getX(), vector.getY(), vector.getZ());
  }

  public void setVectorOnPaper (Vector vector){
    setPointOnPaper(vector.getX(), vector.getY(), vector.getZ());
  }

  public Point snapshot(){
    Point point = new Point();
    point.setValues(isRenewed, vOnPaper, vInReal, type);

    return point;
  }

  void setValues( boolean isRenewed
      , Vector vOnPaper
      , Vector vInReal
      , int type  ){
    this.isRenewed = isRenewed;

    this.vOnPaper.setXYZ(vOnPaper);
    this.vInReal.setXYZ(vInReal);

    this.type = type;

  }
}
```

```java
package com.drancom.programmableMatter.folding.controller.paper;

public class NoLineException extends Exception {

}
```

```java
package com.drancom.programmableMatter.folding.controller.paper;

public class EnergyFunction {

  public static float getGlobalEnergy(Paper paper) {
    float energy = 0.0f;
    int i, j;

    for(i = 0 ; i<paper.numberOfPoints ; i++) {
      for(j = i + 1; j < paper.numberOfPoints ; j++) {
        energy += (paper.pointsOnPaper[i].getXInReal()  - paper.
              pointsOnPaper[j].getXInReal()) * (paper.pointsOnPaper[i].
              getXInReal() - paper.pointsOnPaper[j].getXInReal())
          +  (paper.pointsOnPaper[i].getYInReal()   - paper.pointsOnPaper[j
               ].getYInReal()) * (paper.pointsOnPaper[i].getYInReal() -
               paper.pointsOnPaper[j].getYInReal())
          +  (paper.pointsOnPaper[i].getZInReal()   - paper.pointsOnPaper[j
               ].getZInReal()) * (paper.pointsOnPaper[i].getZInReal() -
               paper.pointsOnPaper[j].getZInReal());
        energy -= (paper.pointsOnPaper[i].getXOnPaper() - paper.
            pointsOnPaper[j].getXOnPaper()) * (paper.pointsOnPaper[i].
            getXOnPaper() - paper.pointsOnPaper[j].getXOnPaper())
          + (paper.pointsOnPaper[i].getYOnPaper()  - paper.pointsOnPaper
              [j].getYOnPaper()) * (paper.pointsOnPaper[i].getYOnPaper()
              - paper.pointsOnPaper[j].getYOnPaper())
          + (paper.pointsOnPaper[i].getZOnPaper()  - paper.pointsOnPaper
              [j].getZOnPaper()) * (paper.pointsOnPaper[i].getZOnPaper()
              - paper.pointsOnPaper[j].getZOnPaper());
      }
    }
    energy = energy * energy;
//    energy = (float) Math.exp(energy) - 1.0f ;

    System.out.printf("Energy = %f" , energy);
    return energy;
  }
}
```

```
package com.drancom.programmableMatter.folding.controller.paper;

public class UnfoldingPaper extends Paper {
  public boolean unfoldingEdge(int EdgeId, int polygonId, float stepAngle)
      {
    Paper paper = this.snapshot();



    return true;
  }





  public UnfoldingPaper snapshot() {
    int i;
    UnfoldingPaper paper = new UnfoldingPaper ();

    Point[] points = new Point [numberOfPoints];
    for(i=0; i<this.numberOfPoints; i++) {
      points[i] = this.pointsOnPaper[i].snapshot();
    }

    Line[] lines = new Line [numberOfLines];
    for(i=0; i<this.numberOfLines; i++) {
      lines[i] = this.lines[i].snapshot(paper, points);
    }

    Polygon[] polygons = new Polygon[numberOfPolygons];
    for(i=0; i<this.numberOfPolygons; i++) {
      polygons[i] = this.polygons[i].snapshot(paper, points);
    }

    paper.setValue(points, numberOfPoints, lines, numberOfLines,polygons,
        numberOfPolygons);

    return paper;

  }

}
```

```
package com.drancom.programmableMatter.folding.controller.paper.util;

import com.wolfram.jlink.KernelLink;
import com.wolfram.jlink.MathLinkException;
import com.wolfram.jlink.MathLinkFactory;

public class Mathematica {
  static KernelLink ml = null;
  static int users = 0;

  public KernelLink load() {
    try {
      ml = MathLinkFactory.createKernelLink("-linkmode launch -linkname 'c
          :\\program files\\wolfram research\\mathematica\\6.0\\mathkernel
          .exe'");
      // Get rid of the initial InputNamePacket the kernel will send
      // when it is launched.
      ml.discardAnswer();
    } catch (MathLinkException e) {
      System.out.println("Fatal error opening link: " + e.getMessage());
      return null;
    }


    return ml;
  }
  public KernelLink getKernelLink(){

    return ml;
  }

  public void close() {
    users--;

  }




}
```

```
package com.drancom.programmableMatter.folding.controller.paper.util;

import com.drancom.programmableMatter.folding.controller.paper.Point;
import com.wolfram.jlink.Expr;
import com.wolfram.jlink.KernelLink;
import com.wolfram.jlink.MathLinkException;
import com.wolfram.jlink.MathLinkFactory;

import quicktime.qd3d.math.Vector3D;
import sun.awt.GlobalCursorManager;
import sun.reflect.ReflectionFactory.GetReflectionFactoryAction;

public class Vector {

//  final static int PLACE_OF_DECIMALS = 10;
  final static float NUMBER_FOR_ROUNDING = 0.5f;

  private float tenPowOfPLACE_OF_DECIMALS; // ten Power Of
      PLACE_OF_DECIMALS

  private float x;
  private float y;
  private float z;

  private float r;
  private float theta;
  private float phi;

  public Vector() {
//    tenPowOfPLACE_OF_DECIMALS = Math.pow(10, PLACE_OF_DECIMALS);
  }

  public float getX(){
    return x;
  }
  public float getY(){
    return y;
  }
  public float getZ(){
    return z;
  }

  public float getR(){
    return r;
  }
  public float getTheta(){
    return theta;
  }
  public float getPhi(){
    return phi;
  }

  public float getLength(){


    return r;
  }

  public void setXYZ(Vector vector){
    setXYZ(vector.getX(), vector.getY(), vector.getZ());
  }

  public void setXYZ(float[] vectorPoints){
    setXYZ(vectorPoints[0], vectorPoints[1], vectorPoints[2]);
  }

  public void setXYZ(float x, float y, float z){

    float[]  vectorTransformed;

/**/
    this.x = x;
    this.y = y;
    this.z = z;
/** /
    this.x  = ((float)(int)(( x * tenPowOfPLACE_OF_DECIMALS)+
          NUMBER_FOR_ROUNDING))/ tenPowOfPLACE_OF_DECIMALS;
    this.y  = ((float)(int)(( y * tenPowOfPLACE_OF_DECIMALS)+
          NUMBER_FOR_ROUNDING))/ tenPowOfPLACE_OF_DECIMALS;
    this.z  = ((float)(int)(( z * tenPowOfPLACE_OF_DECIMALS)+
          NUMBER_FOR_ROUNDING))/ tenPowOfPLACE_OF_DECIMALS;
/**/
    vectorTransformed = transformXYZtoRThetaPhi(x, y, z);
/**/
    r      = vectorTransformed[0];
    theta   = vectorTransformed[1];
    phi    = vectorTransformed[2];
/** /
    r      = ((float)(int)(( vectorTransformed[0] *
          tenPowOfPLACE_OF_DECIMALS)+NUMBER_FOR_ROUNDING))/
          tenPowOfPLACE_OF_DECIMALS;
    theta   = ((float)(int)(( vectorTransformed[1] *
          tenPowOfPLACE_OF_DECIMALS)+NUMBER_FOR_ROUNDING))/
          tenPowOfPLACE_OF_DECIMALS;
    phi    = ((float)(int)(( vectorTransformed[2] *
          tenPowOfPLACE_OF_DECIMALS)+NUMBER_FOR_ROUNDING))/
          tenPowOfPLACE_OF_DECIMALS;
/**/
  }

  public void setRThetaPhi(float[] vector){
    setRThetaPhi(vector[0], vector[1], vector[2]);
  }

  public void setRThetaPhi(float r, float theta, float phi){

    float[]  vectorTransformed;
/**/
    this.r     = r     ;
    this.theta = theta ;
    this.phi   = phi   ;
/** /
    this.r     = ((float)(int)(( r     * tenPowOfPLACE_OF_DECIMALS)+
          NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
    this.theta = ((float)(int)(( theta * tenPowOfPLACE_OF_DECIMALS)+
          NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
    this.phi   = ((float)(int)(( phi   * tenPowOfPLACE_OF_DECIMALS)+
          NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;

/**/
    vectorTransformed = transformRThetaPhitoXYZ(r, theta, phi);
/**/
    this.x = vectorTransformed[0];
    y = vectorTransformed[1];
    z = vectorTransformed[2];
/** /
    x = ((float)(int)(( vectorTransformed[0] * tenPowOfPLACE_OF_DECIMALS)+
          NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
    y = ((float)(int)(( vectorTransformed[1] * tenPowOfPLACE_OF_DECIMALS)+
          NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
    z = ((float)(int)(( vectorTransformed[2] * tenPowOfPLACE_OF_DECIMALS)+
          NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
/**/
  }
  public static double asin(double a) {
    if (a>1) {
      a=1;
    } else if (a<-1) {
      a=-1;
    }

    return Math.asin(a);
  }
  public static double acos(double a) {
    if (a>1) {
      a=1;
    } else if (a<-1) {
      a=-1;
    }


    return Math.acos(a);
  }


  public static float[] transformXYZtoRThetaPhi(float x, float y, float z){

    float[]  vectorTransformed;

    vectorTransformed = new float[3]; // r Theta Phi
    float r, theta, phi;
    float a;

    r = (float) Math.sqrt( ((float) x * (float) x )+ ((float) y * (float) y
          ) + ((float) z * (float) z));  // r
    phi =0;
    theta=0;
    a = 0;

    if( r == 0) {
      phi =0;
      theta =0;

    } else {

      phi  = (float) acos( (double) (z / r) );// Phi   r cos Phi = z   acos
          z/r

      a = (float) (x / (r * Math.sin((float) phi)));
      if (a>1.0){
        a=1;
      } else if (a<-1.0) {
        a=-1;
      }
      theta = (float) acos(a);  // Theta r sin phi * cos theta = x
                                                           // theta = acos
                                                                (( x / r ) /
                                                                 sin phi )
  //     vectorTransformed[1] = (float) Math.atan((float) (y / x));
           // Theta tan Theta = y / x
    }

    vectorTransformed[0] = (float) r;
    vectorTransformed[1] = (float) theta;
    vectorTransformed[2] = (float) phi;

    return vectorTransformed;
  }

  public static float[] transformRThetaPhitoXYZ(float r, float theta, float
        phi){
    float[]  vectorTransformed;
    vectorTransformed = new float[3];
    if (r==0) {
      vectorTransformed[0] = 0.0f;  // x
      vectorTransformed[1] = 0.0f;  // y
      vectorTransformed[2] = 0.0f;                  // z
    } else {
      vectorTransformed[0] = (float)( r * Math.sin((float) phi) * Math.cos
           ((float) theta));   // x
      vectorTransformed[1] = (float)( r * Math.sin((float) phi) * Math.sin
           ((float) theta));   // y
      vectorTransformed[2] = (float)( r * Math.cos((float) phi));
                  // z
    }
    return vectorTransformed;
  }
  public Vector getUnitVector(){
    Vector unitVector = new Vector();
    float x, y, z;
    x = this.x / this.r;
    y = this.y / this.r;
    z = this.z / this.r;

    unitVector.setXYZ(x, y, z);
    return unitVector;
  }

  public void addVector(Vector startPointOfShardlineVector) {
    transform(this, startPointOfShardlineVector);
  }

  public void subtractionVector(Vector vector) {
    Vector invertVector = new Vector();

    invertVector.setXYZ(vector.getX(), vector.getY(), vector.getZ());
    invertVector.invert();

    transform(this, invertVector);
  }

  public void invert() {
    invert(this);
  }



  public void transform(Vector transformVector) {
    transform(this, transformVector);
  }
  public void scale(float scale){
    scale(this, scale);
  }

  public void scale(Vector scaleVector){
    scale(this, scaleVector);
  }

  public float dot(Vector vector){
    return dot(this, vector);
  }

  public void closs(Vector vector) {
    closs(this, vector);
  }


  public void rotationX(float angle){
    rotationX(this, angle);
  }

  public void rotationY(float angle){
    rotationY(this, angle);

  }
  public void rotationZ(float angle){
    rotationZ(this, angle);
  }
  public void rotation(Vector axisVector, float angle){
    rotation(this, axisVector, angle);
  }

  public static void invert(Vector vector) {
    vector.setXYZ(-1 * vector.x, -1 * vector.y, -1 * vector.z);

  }

  public static void transform(Vector vector, Vector transformVector){
    float x, y, z;

    x= vector.getX() + transformVector.getX();
    y= vector.getY() + transformVector.getY();
    z= vector.getZ() + transformVector.getZ();

    vector.setXYZ(x,y,z);
  }

  public static void rotationX(Vector vector, float angle){
    float x, y, z;
  /**/
    x = (float) ( 1                 *  vector.x
            + 0                        *  vector.y
            + 0                  *  vector.z);

    y = (float) ( 0                     *  vector.x
            +      Math.cos((double) angle) *  vector.y
            + -1 * Math.sin((double) angle) *  vector.z);

    z = (float) ( 0                     *  vector.x
            + Math.sin((double) angle)    *  vector.y
            + Math.cos((double) angle)    *  vector.z);
    vector.setXYZ(x, y, z);

  }

  public static void rotationY(Vector vector, float angle){
    float x, y, z;

    x = (float) (     Math.cos((double) angle)    *  vector.x
            + 0                              *  vector.y
            + Math.sin((double) angle)    *  vector.z);

    y = (float) ( 0                      *  vector.x
            + 1                              *  vector.y
            + 0                    *  vector.z);

    z = (float) (-1 * Math.sin((double) angle)    *  vector.x
            + 0                           *  vector.y
            + Math.cos((double) angle)    *  vector.z);

    vector.setXYZ(x, y, z);

  }
  public static void rotationZ(Vector vector, float angle){
    float x, y, z;

    x = (float) ((double) Math.cos((double)     angle)    *  vector.x
            + -1 * (double) Math.sin((double) angle) *  vector.y
            + 0                         *  vector.z);

    y = (float) ((double) Math.sin((double) angle     *  vector.x
            + (double) Math.cos((double) angle))    *  vector.y
            + 0                         *  vector.z);

    z = (float) ( 0                          *  vector.x
            + 0                                  *  vector.y
            + 1                         *  vector.z);

    vector.setXYZ(x, y, z);
  }

  public static void rotation(Vector vector, Vector axisVector, float angle
      ){
    /**
     *  1 + (1-cos(angle))*(x*x-1) -z*sin(angle)+(1-cos(angle))*x*y y*sin(
          angle)+(1-cos(angle))*x*z
     *  z*sin(angle)+(1-cos(angle))*x*y 1 + (1-cos(angle))*(y*y-1) -x*sin(
          angle)+(1-cos(angle))*y*z
     * -y*sin(angle)+(1-cos(angle))*x*z x*sin(angle)+(1-cos(angle))*y*z 1 +
           (1-cos(angle))*(z*z-1)
     */

    float x, y, z;
    float rOnAxis, thetaOnAxis, phiOnAxis;
    float r, theta, pi;
    Vector unitVector;

    unitVector = axisVector.getUnitVector();


/**/ // rotation func 1
    x = (float) ((( 1 + (1 - Math.cos(angle)) * (unitVector.x * unitVector.
         x - 1))                          * vector.x)
        +    (( -1 * unitVector.z * Math.sin(angle) + (1 - Math.cos(angle))
              * unitVector.x * unitVector.y)   * vector.y)
        +    ((        unitVector.y * Math.sin(angle) + (1 - Math.cos(angle))
              * unitVector.x * unitVector.z)   * vector.z));

    y = (float) ((( unitVector.z * Math.sin(angle) + (1 - Math.cos(angle))
         * unitVector.x * unitVector.y)      * vector.x)
        +  (( 1 + (1 - Math.cos(angle)) * (unitVector.y * unitVector.y - 1)
             )                        * vector.y)
        +  ((-1 * unitVector.x * Math.sin(angle) + (1 - Math.cos(angle)) *
             unitVector.y * unitVector.z) * vector.z));

    z = (float) (((-1 * unitVector.y * Math.sin(angle) + (1 - Math.cos(
         angle)) * unitVector.x * unitVector.z) * vector.x)
        +  ((    unitVector.x * Math.sin(angle) + (1 - Math.cos(angle)) *
             unitVector.y * unitVector.z) * vector.y)
        +     (( 1 + (1 - Math.cos(angle)) * (unitVector.z * unitVector.z -
             1))                      * vector.z));
    vector.setXYZ(x, y, z);

/** /  // rotation func 2
    rOnAxis = axisVector.getR();
    thetaOnAxis = axisVector.getTheta();
    phiOnAxis = axisVector.getPhi();

    r = vector.getR();
    theta = vector.getTheta();
    pi = vector.getPhi();


    rotationZ(vector,  1 * thetaOnAxis);
    rotationY(vector,  1 * phiOnAxis);

    rotationX(vector, angle);

    rotationY(vector, -1 * phiOnAxis);
    rotationZ(vector, -1 * thetaOnAxis);

/**/
  }

  public static float dot(Vector vector0, Vector vector1){
    return vector0.getX() * vector1.getX()
      + vector0.getY() * vector1.getY()
      + vector0.getZ() * vector1.getZ();
  }

  public static Vector closs (Vector vector0, Vector vector1) {
    float x0,y0,z0;
    float x1,y1,z1;

    Vector vector = new Vector();

    x0 = vector0.getX();
    y0 = vector0.getY();
    z0 = vector0.getZ();

    x1 = vector1.getX();
    y1 = vector1.getY();
    z1 = vector1.getZ();

    vector.setXYZ((y0*z1 - z0*y1), -1 * (x0 * z1 - z0 * x1 ),  (x0 * y1 -
         y0 * x1));

    return vector;
  }


  public static void scale(Vector vector, float scale){
    vector.setXYZ(scale * vector.getX()
        , scale * vector.getY()
        , scale * vector.getZ());
    }

```

```
public static void scale(Vector vector, Vector scaleVector){
  scale(vector, scaleVector.getR());
  }



public static Vector [] getVectorFrom3Vector(Vector v1, float l1,  //
     length between point1 and point4
    Vector v2, float l2, // length between point2 and point4
    Vector v3, float l3){ // length between point3 and point4

  int i;
  int j;

  // Initiation vectorTemp
  Vector [] vectorTemp = new Vector[3];
  for (i=0 ; i <3; i++) {
    vectorTemp[i] = new Vector();
  }

  Vector vForTransforming = new Vector();


  // Initiation v4
  Vector [] v4 = new Vector [2];
  for (i=0 ; i <2; i++) {
    v4[i] = new Vector();
  }

  vectorTemp[0].setXYZ(v1);
  vectorTemp[1].setXYZ(v2);
  vectorTemp[2].setXYZ(v3);

  float a1, b1, c1;
  float a2, b2, c2;
  float a3, b3, c3;

  a1 = vectorTemp[0].getX();
  b1 = vectorTemp[0].getY();
  c1 = vectorTemp[0].getZ();

  a2 = vectorTemp[1].getX();
  b2 = vectorTemp[1].getY();
  c2 = vectorTemp[1].getZ();

  a3 = vectorTemp[2].getX();
  b3 = vectorTemp[2].getY();
  c3 = vectorTemp[2].getZ();


  String EQ1 = String.format("EQ1:=(x- (%15f))^2+(y- (%15f))^2+(z- (%15f)
      )^2- %15f^2==0",a1,b1,c1,l1 );
  String EQ2 = String.format("EQ2:=(x- (%15f))^2+(y- (%15f))^2+(z- (%15f)
      )^2- %15f^2==0",a2,b2,c2,l2 );
  String EQ3 = String.format("EQ3:=(x- (%15f))^2+(y- (%15f))^2+(z- (%15f)
      )^2- %15f^2==0",a3,b3,c3,l3 );
  String Solve = String.format("NSolve[{EQ1,EQ2,EQ3},{x,y,z}]");

  Mathematica mathematica= new Mathematica();

  KernelLink ml = mathematica.getKernelLink();

  try {

    ml.evaluate(EQ1);
    ml.discardAnswer();
    ml.evaluate(EQ2);
    ml.discardAnswer();
    ml.evaluate(EQ3);
    ml.discardAnswer();

    String result ;
    result = ml.evaluateToOutputForm(Solve, 0);
//      System.out.println(result);

    setVectorFromStringOfResultFromMathmatica(v4, result);


  } catch (MathLinkException e) {
    System.out.println("MathLinkException occurred: " + e.getMessage());
  } finally {
  }

  return v4;
}


private static boolean setVectorFromStringOfResultFromMathmatica(Vector[]
     v4,
    String result) {

  float [] x = new float [2];
  float [] y = new float [2];
  float [] z = new float [2];
  int exponentialNumber[] = new int[6] ;


  int indexOfX=0;
  int indexOfY=0;
  int indexOfZ=0;

  int indexOfExponentialNumber = 0;
  int numberOfExponentialNumber = 0;


  int i;


  String tokenFirstSplit[];
  String tokenFirstLine[];
  String tokenSecondLine[];

      //read command

      result = result.replace('}', ' ');
      result = result.replace('{', ' ');
      result = result.replaceAll("  ", " ");
      result = result.replaceAll(",", " ");
      result = result.trim();

      try {

        tokenFirstSplit = result.split("\n");

      } catch (NullPointerException e) {

        return false;
      }

      if (tokenFirstSplit.length == 2) {
        try {
          tokenFirstLine = tokenFirstSplit[0].split(" ");
          tokenSecondLine = tokenFirstSplit[1].split(" ");
          } catch (NullPointerException e) {
            return false;
          }

        for (i=0; i<tokenFirstLine.length; i++) {

              if (tokenFirstLine[i].equals("")) {

              } else if (0 != Integer.parseInt(tokenFirstLine[i])) {
                if(indexOfExponentialNumber >= 5){
                  int t=0;
                }
                  exponentialNumber[indexOfExponentialNumber]
                                    = Integer.parseInt(tokenFirstLine[i]);
                  indexOfExponentialNumber++;
                  numberOfExponentialNumber++;
                }
            }

          } else {
            try {
              tokenSecondLine = tokenFirstSplit[0].split(" ");
            } catch (NullPointerException e) {
              return false;
            }

          }

          indexOfExponentialNumber=0;

          try {
//            System.out.format(result);

          for (i=0; i<tokenSecondLine.length; i++) {
//            System.out.format("%d\n", i);

            if (tokenSecondLine [i].equals("x")){
              x[indexOfX] = Float.parseFloat(tokenSecondLine[i+2]);


              if (i+3< tokenSecondLine.length
                  && tokenSecondLine [i+3].equals("10")) {
                x[indexOfX]*=Math.pow(10, exponentialNumber[
                     indexOfExponentialNumber]);
                indexOfExponentialNumber++;
              }
              indexOfX++;
              } else if (tokenSecondLine [i].equals("y")){
              y[indexOfY] = Float.parseFloat(tokenSecondLine[i+2]);

              if (i+3< tokenSecondLine.length
                  && tokenSecondLine [i+3].equals("10")) {
                y[indexOfY]*=Math.pow(10, exponentialNumber[
                     indexOfExponentialNumber]);
                indexOfExponentialNumber++;
              }
              indexOfY++;

              } else if (tokenSecondLine [i].equals("z")){
              z[indexOfZ] = Float.parseFloat(tokenSecondLine[i+2]);

              if (i+3 < tokenSecondLine.length
                  && tokenSecondLine [i+3].equals("10")) {
                z[indexOfZ]*=Math.pow(10, exponentialNumber[
                     indexOfExponentialNumber]);
                indexOfExponentialNumber++;
              }
              indexOfZ++;
              }

          }
          }catch (Exception e) {
          e.printStackTrace();
        }

        v4[0].setXYZ((float) x[0],(float) y[0],(float) z[0]);
    v4[1].setXYZ((float) x[1],(float) y[1],(float) z[1]);

    return true;
  }

}
```

```
package com.drancom.programmableMatter.folding.controller;

public class Controller {
  void run() {

  }
}
```

```
package com.drancom.programmableMatter.folding.dataFile;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import com.drancom.programmableMatter.folding.controller.paper.Paper;

public class FileAngleData {
  Paper[] papers;
  String fileName;

  public void build(String fileName, Paper[] papers){
    int i;
    int j;

    int numberOfLines;

    File file = new File(fileName);
    try {
      boolean success = file.createNewFile();
      if (success) {
        // File did not exist and was created
      } else {

      }

    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    String bufferLine;

    this.fileName = fileName;
    this.papers = papers;

    FileOutputStream fileOutputStream = null;

    try {
      fileOutputStream = new FileOutputStream(file);
    } catch (FileNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    numberOfLines = papers[0].getNumberOfEdges();

    for (i=0; i<numberOfLines; i++){
      bufferLine = "";
      bufferLine += papers[0].getLine(i).getAngle();
      for (j=1; j<papers.length; j++) {
        bufferLine += ", ";
        bufferLine += papers[j].getLine(i).getAngle();
      }

      bufferLine += String.format("\n");

      try {
        fileOutputStream.write(bufferLine.getBytes());
      } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }
    }

    try {
      fileOutputStream.close();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

  }
}
```

```
package com.drancom.programmableMatter.folding.dataFile;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;


import com.drancom.programmableMatter.folding.controller.paper.Line;
import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.Point;

public class FileObj {

  Paper paper;
  String fileName;

  public void load(String fileName, Paper paper){
    File file = new File(fileName);

    this.fileName = fileName;
    this.paper = paper;

    FileInputStream fileInputStream = null;
        BufferedReader bufferedReader = null;

        int index_v  = 0;
        int index_vt = 0;
        int index_e  = 0;

        String [] token ;


        String bufferLine;

        if(file.exists()) {

          try {
            fileInputStream = new FileInputStream(file);
            } catch (Exception e) {
              e.printStackTrace();
            }

            bufferedReader = new BufferedReader(new InputStreamReader(
                 fileInputStream));
            while(true) {
              // Load buffer
              try {
                bufferLine = bufferedReader.readLine();
              } catch (IOException e) {
                e.printStackTrace();
            break;
              }

              //read command
              try {
                token = bufferLine.split(" ");
              } catch (NullPointerException e) {

                paper.build();

                return;
              }

              if (token[0].equals("vt")){

                Point temp_point;


                temp_point = paper.getPoint(index_vt);

                if (temp_point == null)
                {
                  temp_point = new Point();
                }

                temp_point.setPointOnPaper( Float.parseFloat(token[1]),
                              Float.parseFloat(token[2]),
                              0.0f);

                paper.setPoint(index_vt,temp_point);

                index_vt++;

              } else if (token[0].equals("#e")){

              Line temp_line;
              Point temp_startPoint;
              Point temp_endPoint;
              int index_startPoint;
              int index_endPoint   ;
              float temp_angle;
              int temp_typeOfLine;

              temp_line = paper.getLine(index_e);

              if (temp_line == null)
              {
                temp_line = new Line();
              }

              index_startPoint = Integer.parseInt(token[1]) - 1;
              index_endPoint   = Integer.parseInt(token[2]) - 1;

              temp_startPoint = paper.getPoint(index_startPoint);

              temp_endPoint = paper.getPoint(index_endPoint);

              temp_typeOfLine = Integer.parseInt(token[3]);
```

```java

                /**
                 * TYPE_STATIC_LINE    = 0;
                 * TYPE_EDGE_LINE      = 1;
                 * TYPE_POSITIVE_LINE  = 2;
                 * TYPE_NEGATIVE_LINE  = 3;
                 */


                temp_angle      = Float.parseFloat(token[4]);

                if (temp_typeOfLine == Line.TYPE_POSITIVE_LINE) {
                  temp_angle = (float) ( 1 * (Math.PI / 180) * temp_angle);
                } else if (temp_typeOfLine == Line.TYPE_NEGATIVE_LINE)
                {
                  temp_angle = (float) ( -1 * (Math.PI / 180) * temp_angle);
                } else {
                  temp_angle = (float) ( 1 * (Math.PI / 180) * temp_angle);
                }


                temp_line.setLine(paper,
                        temp_startPoint,
                        temp_endPoint,
                        temp_angle,
                        temp_typeOfLine);

                paper.setLine(index_e, temp_line);

                index_e++;

                }


            }
        }
  }
}
```

```java
package com.drancom.programmableMatter.folding.dataFile;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.origami.planner.Plan;

public interface FilePlan {

  public void build(String fileName, Plan plan);
  public void build(String fileName, Paper[] papers);
  public void build(String fileName, Plan plan, Paper[] papers);
  public void read (String fileName, Plan plan);
  public void read (String fileName, Paper[] papers);
  public void read (String fileName, Plan plan, Paper[] papers);

}
```

```java
package com.drancom.programmableMatter.folding.dataFile;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.origami.planner.Plan;

public class FilePlanForAngleActuration implements FilePlan{
  Paper[] papers;
  String fileName;

  public void build(String fileName, Paper[] papers){
    int i;
    int time;
    int j;
    int k;
    int l;

    boolean ready;

    int numberOfLines;

    File file = new File(fileName);

    try {
      boolean success = file.createNewFile();
      if (success) {
        // File did not exist and was created
      } else {
      }

    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    String bufferLine = new String();
    String bufferTime = new String();
    String bufferLevel00 = new String();
    String bufferLevel01 = new String();
    String bufferLevel02 = new String();
    String bufferLevel03 = new String();
    String bufferLevel04 = new String();

    String changeLine = String.format("\n");

    this.fileName = fileName;
    this.papers = papers;

    FileOutputStream fileOutputStream = null;

    try {
      fileOutputStream = new FileOutputStream(file);
    } catch (FileNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    numberOfLines = papers[0].getNumberOfEdges();

    // read paper
    // set up the level.
    // record the levels of edges when the level is change.

    bufferLine = "time, level 1, level 2,  level -1, level -2, level  0" +
        changeLine;

    try {
      fileOutputStream.write(bufferLine.getBytes());
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    time = 0;
    l = papers.length -1;
    ready = false;
    for (i = papers.length -1; i > 0; i-- ) {
      l = i;
```

```
        for (j = 0; j < numberOfLines ; j++ ) {
          if (papers[i     ].getLine(j).getLevelOfActuratingPower()!= 0.0f) {
            ready = true;
            break;
          }
        }

        if (ready){
          break;

        }
      }

      bufferTime = Integer.toString(i);
      bufferLevel00 = "";
      bufferLevel01 = "";
      bufferLevel02 = "";
      bufferLevel03 = "";
      bufferLevel04 = "";

      for (k = 0; k < numberOfLines ; k++) {

        if (papers[i].getLine(k).getLevelOfActuratingPower() == 2f) {
          bufferLevel02 += "f" + k + " ";
        } else if (papers[i].getLine(k).getLevelOfActuratingPower() == -2f) {
          bufferLevel04 += "f" + k + " ";
        } else if (papers[i].getLine(k).getLevelOfActuratingPower() > 0.0f) {
          bufferLevel01 += "f" + k + " ";
        } else if (papers[i].getLine(k).getLevelOfActuratingPower() < 0.0f) {
          bufferLevel03 += "f" + k + " ";
        } else {
          bufferLevel00 += "f" + k + " ";
        }
      }

      bufferTime = Integer.toString(time);

      bufferLine = bufferTime + ", ";
      bufferLine += bufferLevel01 + ", ";
      bufferLine += bufferLevel02 + ", ";
      bufferLine += bufferLevel03 + ", ";
      bufferLine += bufferLevel04 + ", ";
//      bufferLine += bufferLevel00;
      bufferLine += changeLine;

      try {
        fileOutputStream.write(bufferLine.getBytes());
      } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }


      time++;

      for (i = l-1; i > 1; i-- ) {

        for (j = 0; j < numberOfLines ; j++ ) {
          if (Math.abs(papers[i     ].getLine(j).getLevelOfActuratingPower()) !=
            Math.abs(papers[i + 1].getLine(j).getLevelOfActuratingPower()) ) {
            // lavel  1, lavel  2, lavel -1, lavel -2

            bufferTime = Integer.toString(i);
            bufferLevel00 = "";
            bufferLevel01 = "";
            bufferLevel02 = "";
            bufferLevel03 = "";
            bufferLevel04 = "";

            for (k = 0; k < numberOfLines ; k++) {


              if (papers[i].getLine(k).getLevelOfActuratingPower() == 2f) {
                bufferLevel02 += "f" + k + " ";
              } else if (papers[i].getLine(k).getLevelOfActuratingPower() == -2f) {
                bufferLevel04 += "f" + k + " ";
              } else if (papers[i].getLine(k).getLevelOfActuratingPower() > 0.0f) {
                bufferLevel01 += "f" + k + " ";
              } else if (papers[i].getLine(k).getLevelOfActuratingPower() < 0.0f) {
                bufferLevel03 += "f" + k + " ";
              } else {
                bufferLevel00 += "f" + k + " ";
              }
            }


            bufferTime = Integer.toString(time);

            bufferLine = bufferTime + ", ";
            bufferLine += bufferLevel01 + ", ";
            bufferLine += bufferLevel02 + ", ";
            bufferLine += bufferLevel03 + ", ";
            bufferLine += bufferLevel04 + ", ";
//            bufferLine += bufferLevel00;
            bufferLine += changeLine;

            try {
              fileOutputStream.write(bufferLine.getBytes());
            } catch (IOException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
            }
            break;
          }
        }
        time++;
      }

      try {
        fileOutputStream.close();
      } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }
    }

    @Override
    public void build(String fileName, Plan plan) {
      // TODO Auto-generated method stub

    }

    @Override
    public void read(String fileName, Plan plan) {
      // TODO Auto-generated method stub

    }

    @Override
    public void read(String fileName, Paper[] papers) {
      // TODO Auto-generated method stub

    }
```

```

  @Override
  public void build(String fileName, Plan plan, Paper[] papers) {
    // TODO Auto-generated method stub

  }

  @Override
  public void read(String fileName, Plan plan, Paper[] papers) {
    // TODO Auto-generated method stub

  }
}
```

```
package com.drancom.programmableMatter.folding.dataFile;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.origami.planner.Plan;
import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
    ;

public class FilePlanForWiring implements FilePlan {


  public void build(String fileName, Plan plan){
    int i;
    int j;
    int k;
    int l;

    int time;

    boolean ready;

    int numberOfLines;

    File file = new File(fileName);
    PlanForWiring planForWiring = (PlanForWiring) plan;
    boolean [][][][] planTable = planForWiring.getPlanTable();

    try {
      boolean success = file.createNewFile();
      if (success) {
        // File did not exist and was created
      } else {
      }

    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    String bufferTitle = new String();
    String bufferLine  = new String();
    String changeLine = String.format("\n");

    FileOutputStream fileOutputStream = null;

    try {
      fileOutputStream = new FileOutputStream(file);
    } catch (FileNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    // read paper
    // set up the level.
    // record the levels of edges when the level is change.

    bufferTitle = "[folding][phases][inside][edgeNumber][activation]";
    bufferLine = bufferTitle;
    bufferLine += changeLine;

    try {
      fileOutputStream.write(bufferLine.getBytes());
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    // planTable [folding][phases][inside][edgeNumber][activation]
    for (i = 0; i < 2 ; i++) {
      for (j = 0; j < planForWiring.getNumberOfPhases(); j++) {
        for (k = 0; k < 2 ; k++ ) {
          for (l = 0 ; l < planForWiring.getNumberOfEdges(); l++ ) {

            bufferLine = String.format("%d, %d, %d, %d, %s", i, j, k, l,
                Boolean.toString(planTable[i][j][k][l]));
            bufferLine += changeLine;

            // print to file
            try {
              fileOutputStream.write(bufferLine.getBytes());
            } catch (IOException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
            }
          }
        }
      }
    }

    try {
      fileOutputStream.close();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

  @Override
  public void build(String fileName, Paper[] papers) {
    // TODO Auto-generated method stub

  }

  @Override
  public void read(String fileName, Plan plan) {
    // TODO Auto-generated method stub

  }

  @Override
  public void read(String fileName, Paper[] papers) {
    // TODO Auto-generated method stub

  }

  @Override
  public void build(String fileName, Plan plan, Paper[] papers) {
    // TODO Auto-generated method stub

  }
```

```java

  @Override
  public void read(String fileName, Plan plan, Paper[] papers) {
    // TODO Auto-generated method stub

  }
}
```

```java
package com.drancom.programmableMatter.folding.dataFile;

import java.io.File;
import java.io.FileInputStream;

public class FilePmf {
  void load(String fileName){
    File file = new File(fileName);

        if(file.exists()) {
           FileInputStream fileInputStream = null;
           try {
             fileInputStream = new FileInputStream(file);
           } catch (Exception e) {

           }
        }
  }
}
```

```java
package com.drancom.programmableMatter.folding.dataFile;

public class Transform {

}
```

```java
package com.drancom.programmableMatter.folding.monitor;


import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.*;

import javax.media.opengl.*;

import sun.text.normalizer.UProperty;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.Point;
import com.drancom.programmableMatter.folding.controller.paper.Polygon;
import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.sun.opengl.util.*;

public class MainWindow implements GLEventListener, MouseListener,
     MouseMotionListener {
  Paper[] papers;

  // light
  // float pos0[] = {  -100.0f,  130.0f, 150.0f,  1.0f };
  public final static float pos0[] =  {  -100.0f,  100.0f, 100.0f,  1.0f };
  public final static float ambientLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
  public final static float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
  public final static float specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};

  public final static float WRITE[]  = {  1.0f,  1.0f,  1.0f,  0.2f };
  public final static float RED[]    = {  1.0f,  0.0f,  0.0f,  1.0f };
  public final static float GREEN[]  = {  0.0f,  1.0f,  0.0f,  1.0f };
  public final static float YELLOW[] = {  1.0f,  1.0f,  0.0f,  1.0f };
  public final static float BLUE[]   = { 0.0f, 0.0f, 1.0f, 0.0f };
  public final static float BLACK[]  = { 0.0f, 0.0f, 0.0f, 1.0f };

  final static int SPEED_OF_ANIMATION = 15; //10 is default
  int counterForSpeedOfAnimation = 0;

  boolean isAnimating=false;

  public void run(Paper[] papers) {
    this.papers = papers;
    paperGlId = new int[papers.length];

    Frame frame = new Frame("Play Window - Programmable Matter by Folding")
        ;
      GLCanvas canvas = new GLCanvas();

    canvas.addGLEventListener(this);
    frame.add(canvas);
    frame.setSize(800, 800);
    final Animator animator = new Animator(canvas);
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
            // Run this on another thread than the AWT event queue to
        // make sure the call to Animator.stop() completes before
        // exiting
        new Thread(new Runnable() {
          public void run() {
            animator.stop();
            System.exit(0);
          }
        }).start();
          }
    });
      frame.show();
      animator.start();
  }

  private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;

  private int paperGlId[];
  private int currentIndex_PaperGlId;

  private boolean isFolding = true;

  public boolean isFolding() {
    return isFolding;
  }

  private int prevMouseX, prevMouseY;
  private boolean mouseRButtonDown = false;

  public void init(GLAutoDrawable drawable) {
    int i;

    // Use debug pipeline
    // drawable.setGL(new DebugGL(drawable.getGL()));

    GL gl = drawable.getGL();

    System.err.println("INIT GL IS: " + gl.getClass().getName());
    System.err.println("Chosen GLCapabilities: " + drawable.
         getChosenGLCapabilities());

    gl.setSwapInterval(1);

    // Blend
    gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE);
    gl.glEnable(GL.GL_BLEND);


```

```
//    gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
//      gl.glClearDepth(1.0f);

    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos0, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambientLight, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuseLight, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular, 0);

    gl.glEnable(GL.GL_CULL_FACE);

    gl.glEnable(GL.GL_LIGHTING);
      gl.glEnable(GL.GL_LIGHT0);

//    gl.glEnable(GL.GL_DEPTH_TEST);

      /* make the papers */
      for (i=0; i<papers.length; i++){
        paperGlId[i] = gl.glGenLists(1);
        gl.glNewList(paperGlId[i], GL.GL_COMPILE);
        buildGlPaper(gl, papers[i]);
        gl.glEndList();
      }

    // currentIndex_PaperGlId = 0;
      currentIndex_PaperGlId = papers.length -1;

  //  gl.glEnable(GL.GL_NORMALIZE);

      drawable.addMouseListener(this);
      drawable.addMouseMotionListener(this);
  }

  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
    GL gl = drawable.getGL();

    float h = (float)height / (float)width;

    gl.glMatrixMode(GL.GL_PROJECTION);

    System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
    System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
    System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
    gl.glLoadIdentity();
    gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0.0f, 0.0f, -40.0f);
  }

  public void display(GLAutoDrawable drawable) {
    // Turn the gears' teeth

    // Get the GL corresponding to the drawable we are animating
    GL gl = drawable.getGL();

    // Special handling for the case where the GLJPanel is translucent
    // and wants to be composited with other Java 2D content
    if ((drawable instanceof GLJPanel) &&
        !((GLJPanel) drawable).isOpaque() &&
        ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
      gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
    } else {
      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    }


    // Rotate the entire assembly of gears based on how the user
    // dragged the mouse around
    gl.glPushMatrix();
    gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
    gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
    gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);

    // Place the first gear and call its display list
    gl.glPushMatrix();
      gl.glTranslatef(-5.0f, -5.0f, 0.0f);
      gl.glCallList(paperGlId[((int) currentIndex_PaperGlId)]);
    gl.glPopMatrix();


    // Remember that every push needs a pop; this one is paired with
    // rotating the entire gear assembly
    gl.glPopMatrix();
    if (isAnimating == true){
      if (isFolding() == true ){
        counterForSpeedOfAnimation--;

        if (counterForSpeedOfAnimation <= 0) {
          counterForSpeedOfAnimation = SPEED_OF_ANIMATION;

          currentIndex_PaperGlId--;

          if (currentIndex_PaperGlId < 0) {
            currentIndex_PaperGlId = 0;
            isFolding = false;
            isAnimating = false;
          }
        }
      } else {
        counterForSpeedOfAnimation++;
        if (counterForSpeedOfAnimation >= SPEED_OF_ANIMATION) {
          counterForSpeedOfAnimation = 0;

          currentIndex_PaperGlId++;

          if (currentIndex_PaperGlId >= papers.length - 1 ) {
            currentIndex_PaperGlId = papers.length- 1;
            isFolding = true;
            isAnimating = false;
          }
        }
      }
    }
  }

  public void displayChanged(GLAutoDrawable drawable, boolean modeChanged, boolean deviceChanged) {}

  public void buildGlPaper(GL gl, Paper paper){
    int i;
    int j;
    int numberOfLine = paper.getNumberOfEdges();
    int numberOfPolygon = paper.getNumberOfPolygons();

    Polygon polygon;
    Point[] polygonPoints;

    Vector startPointVector;
    Vector endPointVector ;

    gl.glShadeModel(GL.GL_FLAT);

    gl.glNormal3f(0.0f, 0.0f, 1.0f);

    /* draw polygon */
    numberOfPolygon = paper.getNumberOfPolygons();

    for (i=0; i<numberOfPolygon; i++) {
      polygon = paper.getPolygon(i);

      polygonPoints = polygon.getPoints();

      gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WRITE, 0);

      gl.glBegin(GL.GL_TRIANGLES);
        for (j=0; j<3; j++){
          gl.glVertex3d(polygonPoints[j].getXInReal() * 8
              , polygonPoints[j].getYInReal() * 8
              , polygonPoints[j].getZInReal() * 8 );

        }
        for (j=2; j>=0; j--){
        gl.glVertex3d(polygonPoints[j].getXInReal() * 8
              , polygonPoints[j].getYInReal() * 8
              , polygonPoints[j].getZInReal() * 8 );

        }

      gl.glEnd();


    }

    /* draw lines */
    numberOfLine = paper.getNumberOfEdges();

    for (i=0; i<numberOfLine; i++) {

        gl.glLineWidth(1.0f);

      startPointVector = paper.getLine(i).getStartPoint().getVectorInReal()
           ;
      endPointVector = paper.getLine(i).getEndPoint().getVectorInReal();


      if(paper.getLine(i).getLevelOfActuratingPower() > 0.01) {

        // actuating
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, RED, 0);

      }else if (paper.getLine(i).getLevelOfActuratingPower() < -0.01){

        // passive moving
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, BLUE, 0);

      }else {

        // stop
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WRITE, 0);
      }


      gl.glBegin(GL.GL_LINES);

      gl.glVertex3f((float)startPointVector.getX()*8,
          (float)startPointVector.getY()*8,
          (float)startPointVector.getZ()*8);

      gl.glVertex3f((float)endPointVector.getX()*8,
          (float)endPointVector.getY()*8,
          (float)endPointVector.getZ()*8);
      gl.glEnd();


    }
  }

    // Methods required for the implementation of MouseListener
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {
      prevMouseX = e.getX();
      prevMouseY = e.getY();
      if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
        mouseRButtonDown = true;
      }

      if (mouseRButtonDown == true) {
        isAnimating = true;
      }


  }

  public void mouseReleased(MouseEvent e) {
    if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
      mouseRButtonDown = false;
      }

  }

  public void mouseClicked(MouseEvent e) {}

  // Methods required for the implementation of MouseMotionListener
  public void mouseDragged(MouseEvent e) {
    if (mouseRButtonDown == false) {
      int x = e.getX();
      int y = e.getY();
      Dimension size = e.getComponent().getSize();

      float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
      float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);

      prevMouseX = x;
      prevMouseY = y;

      view_rotx += thetaX;
      view_roty += thetaY;
    }else {

    }
  }

  public void mouseMoved(MouseEvent e) {

  }
}
```

```
package com.drancom.programmableMatter.folding.monitor;



import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.media.opengl.*;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.Point;
import com.drancom.programmableMatter.folding.controller.paper.Polygon;
import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
     ;
import com.drancom.programmableMatter.folding.origami.planner.
     PlannerForWiring;
import com.sun.opengl.util.Animator;


public class MainWindowForFoldingRobotWiring  implements GLEventListener,
     MouseListener, MouseMotionListener {
  PlanForWiring planForWiring;
  boolean [][][][] planOfTable;
  Paper[] papers;

  // light
  // float pos0[] =  {  -100.0f,  130.0f, 150.0f,  1.0f };
  float pos0[] =  {  -100.0f,  100.0f, 100.0f,  1.0f };
  float ambientLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
  float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
  float specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};



  float white[] = {  1.0f,  1.0f,  1.0f,  0.2f };
  float red[] =   {  1.0f,  0.0f,  0.0f,  1.0f };
  float green[] = {  0.0f,  1.0f,  0.0f,  1.0f };
  float yellow[] = {  1.0f,  1.0f,  0.0f,  1.0f };
  float blue[] = { 0.0f, 0.0f, 1.0f, 0.0f };
  float black[] = { 0.0f, 0.0f, 0.0f, 1.0f };

  final static int SPEED_OF_ANIMATION = 15; //10 is default
  int counterForSpeedOfAnimation = 0;

  boolean isAnimating=false;

  public void run(Paper[] papers, PlanForWiring planForWiring) {
    this.papers = papers;
    paperGlId = new int[papers.length];

    this.planForWiring = planForWiring;
    planOfTable = planForWiring.getPlanTable();

    Frame frame = new Frame("Play Window - Programmable Matter by Folding")
        ;
      GLCanvas canvas = new GLCanvas();

    canvas.addGLEventListener(this);
    frame.add(canvas);
    frame.setSize(800, 800);
    final Animator animator = new Animator(canvas);
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
            // Run this on another thread than the AWT event queue to
        // make sure the call to Animator.stop() completes before
        // exiting
        new Thread(new Runnable() {
          public void run() {
            animator.stop();
            System.exit(0);
          }
        }).start();
          }
    });
      frame.show();
      animator.start();
  }

  private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;

  private int paperGlId[];
  private int currentIndex_PaperGlId;

  private boolean isFolding = true;

  public boolean isFolding() {
    return isFolding;
  }

  private int prevMouseX, prevMouseY;
  private boolean mouseRButtonDown = false;

  public void init(GLAutoDrawable drawable) {
    int i;

    // Use debug pipeline
    // drawable.setGL(new DebugGL(drawable.getGL()));

    GL gl = drawable.getGL();

    System.err.println("INIT GL IS: " + gl.getClass().getName());
    System.err.println("Chosen GLCapabilities: " + drawable.
          getChosenGLCapabilities());

    gl.setSwapInterval(1);

    // Blend
    gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE);
    gl.glEnable(GL.GL_BLEND);


//     gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
//       gl.glClearDepth(1.0f);

    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos0, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambientLight, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuseLight, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular, 0);

    // gl.glEnable(GL.GL_CULL_FACE);

    gl.glEnable(GL.GL_LIGHTING);
      gl.glEnable(GL.GL_LIGHT0);

//     gl.glEnable(GL.GL_DEPTH_TEST);

      /* make the papers */
      for (i=0; i<papers.length; i++){
        paperGlId[i] = gl.glGenLists(1);
        gl.glNewList(paperGlId[i], GL.GL_COMPILE);
        buildGlPaper(gl, papers[i]);
        gl.glEndList();
      }

    // currentIndex_PaperGlId = 0;
      currentIndex_PaperGlId = papers.length -1;

  //   gl.glEnable(GL.GL_NORMALIZE);

      drawable.addMouseListener(this);
      drawable.addMouseMotionListener(this);
  }

  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
        height) {
    GL gl = drawable.getGL();

    float h = (float)height / (float)width;

    gl.glMatrixMode(GL.GL_PROJECTION);

    System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
    System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
    System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
    gl.glLoadIdentity();
    gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0.0f, 0.0f, -40.0f);
  }

  public void display(GLAutoDrawable drawable) {
    // Turn the gears' teeth

    // Get the GL corresponding to the drawable we are animating
    GL gl = drawable.getGL();

    // Special handling for the case where the GLJPanel is translucent
    // and wants to be composited with other Java 2D content
    if ((drawable instanceof GLJPanel) &&
        !((GLJPanel) drawable).isOpaque() &&
        ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
      gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
    } else {
      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    }


    // Rotate the entire assembly of gears based on how the user
    // dragged the mouse around
    gl.glPushMatrix();
    gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
    gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
    gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);

    // Place the first gear and call its display list
    gl.glPushMatrix();
      gl.glTranslatef(-5.0f, -5.0f, 0.0f);
      gl.glCallList(paperGlId[((int) currentIndex_PaperGlId)]);
    gl.glPopMatrix();


    // Remember that every push needs a pop; this one is paired with
    // rotating the entire gear assembly
    gl.glPopMatrix();
    if (isAnimating == true){
      if (isFolding() == true ){
        counterForSpeedOfAnimation--;

        if (counterForSpeedOfAnimation <= 0) {
          counterForSpeedOfAnimation = SPEED_OF_ANIMATION;

          currentIndex_PaperGlId--;

          if (currentIndex_PaperGlId < 0) {
            currentIndex_PaperGlId = 0;
            isFolding = false;
            isAnimating = false;
          }
        }
      } else {
        counterForSpeedOfAnimation++;
        if (counterForSpeedOfAnimation >= SPEED_OF_ANIMATION) {
          counterForSpeedOfAnimation = 0;

          currentIndex_PaperGlId++;

          if (currentIndex_PaperGlId >= papers.length - 1 ) {
            currentIndex_PaperGlId = papers.length- 1;
            isFolding = true;
            isAnimating = false;
          }
        }
      }
    }
  }

  public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
       boolean deviceChanged) {}

  public void buildGlPaper(GL gl, Paper paper){

    // for phase 0 and 1
    int i;
    int j;
    int numberOfLine = paper.getNumberOfEdges();
    int numberOfPolygon = paper.getNumberOfPolygons();

    Polygon polygon;
    Point[] polygonPoints;

    Vector startPointVector;
    Vector endPointVector ;

    gl.glShadeModel(GL.GL_FLAT);

    gl.glNormal3f(0.0f, 0.0f, 1.0f);

    /* draw polygon */
    numberOfPolygon = paper.getNumberOfPolygons();

    for (i=0; i<numberOfPolygon; i++) {
      polygon = paper.getPolygon(i);

      polygonPoints = polygon.getPoints();

      gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);

      gl.glBegin(GL.GL_TRIANGLES);
        for (j=0; j<3; j++){
          gl.glVertex3d(polygonPoints[j].getXInReal() * 8
              , polygonPoints[j].getYInReal() * 8
              , polygonPoints[j].getZInReal() * 8 );

        }
//        for (j=2; j>=0; j--){
//          gl.glVertex3f(polygonPoints[j].getXInReal() * 8
//              , polygonPoints[j].getYInReal() * 8
//              , polygonPoints[j].getZInReal() * 8 );
//
//        }

      gl.glEnd();


    }

    /* draw lines */
    numberOfLine = paper.getNumberOfEdges();

    for (i=0; i<numberOfLine; i++) {

        gl.glLineWidth(1.0f);

      startPointVector = paper.getLine(i).getStartPoint().getVectorInReal();
      endPointVector = paper.getLine(i).getEndPoint().getVectorInReal();

    /**/

      if(planOfTable[0][0][0][i] == true) {

        // actuating
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, red, 0);

      }else if (planOfTable[0][0][1][i] == true){

        // passive moving
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, blue, 0);

      }else {

        // stop
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);
      }
/** /
      if(paper.getLine(i).getLevelOfActuratingPower() > 0.01) {

        // actuating
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, red, 0);

      }else if (paper.getLine(i).getLevelOfActuratingPower() < -0.01){

        // passive moving
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, blue, 0);

      }else {

        // stop
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);
      }
/**/

      gl.glBegin(GL.GL_LINES);

      gl.glVertex3f((float)startPointVector.getX()*8,
          (float)startPointVector.getY()*8,
          (float)startPointVector.getZ()*8);

      gl.glVertex3f((float)endPointVector.getX()*8,
          (float)endPointVector.getY()*8,
          (float)endPointVector.getZ()*8);
      gl.glEnd();


    }
  }

    // Methods required for the implementation of MouseListener
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {
      prevMouseX = e.getX();
      prevMouseY = e.getY();
      if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
        mouseRButtonDown = true;
      }

      if (mouseRButtonDown == true) {
        isAnimating = true;
      }


  }

  public void mouseReleased(MouseEvent e) {
    if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
      mouseRButtonDown = false;
      }

  }

  public void mouseClicked(MouseEvent e) {}

  // Methods required for the implementation of MouseMotionListener
  public void mouseDragged(MouseEvent e) {
    if (mouseRButtonDown == false) {
      int x = e.getX();
      int y = e.getY();
      Dimension size = e.getComponent().getSize();

      float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
      float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);

      prevMouseX = x;
      prevMouseY = y;

      view_rotx += thetaX;
      view_roty += thetaY;
    }else {

    }
  }

  public void mouseMoved(MouseEvent e) {

  }

}
```

```
package com.drancom.programmableMatter.folding.monitor;



import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.ArrayList;

import javax.media.opengl.*;

import com.drancom.programmableMatter.folding.controller.paper.Line;
import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.Point;
import com.drancom.programmableMatter.folding.controller.paper.Polygon;

import com.sun.opengl.util.Animator;


public class MonitorOfPaperArray implements GLEventListener, MouseListener
     , MouseMotionListener {
  public final static float ZOOM_MAGNIFICATION = 8.0f;
  public final static float LINEWIDTH = 2.0f;

  ArrayList<Paper> paperArray;

  // light
  // float pos0[] = { -100.0f, 130.0f, 150.0f, 1.0f };
  float pos0[] = { -100.0f, 100.0f, 100.0f, 1.0f };
  float ambientLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
  float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
  float specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};

  float white[] = {  1.0f,  1.0f,  1.0f,  0.2f };
  float red[] =   {  1.0f,  0.0f,  0.0f,  1.0f };
  float green[] = {  0.0f,  1.0f,  0.0f,  1.0f };
  float yellow[] = {  1.0f,  1.0f,  0.0f,  1.0f };
  float blue[] = { 0.0f, 0.0f, 1.0f, 0.0f };
  float black[] = { 0.0f, 0.0f, 0.0f, 1.0f };

  final static int SPEED_OF_ANIMATION = 15; //10 is default
  int counterForSpeedOfAnimation = 0;

  boolean isAnimating = false;


  int numberOfPapers;

  Frame frame;



  public void run(ArrayList<Paper> paperArray) {
    numberOfPapers = paperArray.size();

    this.paperArray = paperArray;

    paperGlId = new int[numberOfPapers];

    frame = new Frame("Monitor Of Unfolding Algorithm - Programmable Matter
          by Folding");
      GLCanvas canvas = new GLCanvas();

    canvas.addGLEventListener(this);
    frame.add(canvas);
    frame.setSize(800, 800);
    final Animator animator = new Animator(canvas);
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
            // Run this on another thread than the AWT event queue to
        // make sure the call to Animator.stop() completes before
        // exiting
        new Thread(new Runnable() {
          public void run() {
            animator.stop();
            System.exit(0);
          }
        }).start();
          }
    });
      frame.show();
      animator.start();
  }


  private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;

  private int paperGlId[];
  private int currentIndex_PaperGlId;

  private boolean isFolding = true;

  public boolean isFolding() {
    return isFolding;
  }

  private int prevMouseX, prevMouseY;
  private boolean mouseRButtonDown = false;

  public void init(GLAutoDrawable drawable) {
    int i;

    // Use debug pipeline
    // drawable.setGL(new DebugGL(drawable.getGL()));

    GL gl = drawable.getGL();

    System.err.println("INIT GL IS: " + gl.getClass().getName());
    System.err.println("Chosen GLCapabilities: " + drawable.
         getChosenGLCapabilities());

    gl.setSwapInterval(1);

    // Blend
    gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE);
    gl.glEnable(GL.GL_BLEND);


//    gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
//      gl.glClearDepth(1.0f);

    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos0, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambientLight, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuseLight, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular, 0);

    //gl.glEnable(GL.GL_CULL_FACE);

    gl.glEnable(GL.GL_LIGHTING);
      gl.glEnable(GL.GL_LIGHT0);

//    gl.glEnable(GL.GL_DEPTH_TEST);

      /* make the papers */
      for (i=0; i < numberOfPapers; i++){
        paperGlId[i] = gl.glGenLists(1);
        gl.glNewList(paperGlId[i], GL.GL_COMPILE);
          buildGlPaper(gl, i, paperArray.get(i));
        gl.glEndList();
      }

    currentIndex_PaperGlId = 0;


//    gl.glEnable(GL.GL_NORMALIZE);

      drawable.addMouseListener(this);
      drawable.addMouseMotionListener(this);
  }

  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
        height) {
    GL gl = drawable.getGL();

    float h = (float)height / (float)width;

    gl.glMatrixMode(GL.GL_PROJECTION);

    System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
    System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
    System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
    gl.glLoadIdentity();
    gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0.0f, 0.0f, -40.0f);
  }

  public void display(GLAutoDrawable drawable) {
    // Turn the gears' teeth

    // Get the GL corresponding to the drawable we are animating
    GL gl = drawable.getGL();

    // Special handling for the case where the GLJPanel is translucent
    // and wants to be composited with other Java 2D content
    if ((drawable instanceof GLJPanel) &&
        !((GLJPanel) drawable).isOpaque() &&
        ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
      gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
    } else {
      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    }


    // Rotate the entire assembly of gears based on how the user
    // dragged the mouse around
    gl.glPushMatrix();
    gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
    gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
    gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);

    // Place the first gear and call its display list
    gl.glPushMatrix();
      gl.glTranslatef(-0.0f, -0.0f, 0.0f);
      gl.glCallList(paperGlId[((int) currentIndex_PaperGlId)]);
    gl.glPopMatrix();


    // Remember that every push needs a pop; this one is paired with
    // rotating the entire gear assembly
    gl.glPopMatrix();

  }

  public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
      boolean deviceChanged) {}

  private void buildGlPaper(GL gl, int index, Paper paper) {

    // for phase 0 and 1
    int i;
    int j;
    int numberOfEdges = paper.getNumberOfEdges();
    int numberOfPolygon;

    Polygon[] polygons;
    Point[] polygonPoints;
    Line line;



    gl.glShadeModel(GL.GL_FLAT);
    gl.glNormal3f(0.0f, 0.0f, 1.0f);

    /* draw polygon */
    numberOfPolygon = paper.getNumberOfPolygons();
    polygons = paper.getPolygons();

    for (i=0; i<numberOfPolygon; i++) {

      polygonPoints = polygons[i].getPoints();

      gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);

      gl.glBegin(GL.GL_TRIANGLES);
        for (j=0; j<3; j++){
          gl.glVertex3d(polygonPoints[j].getXInReal() * ZOOM_MAGNIFICATION
                , polygonPoints[j].getYInReal() * ZOOM_MAGNIFICATION
                , polygonPoints[j].getZInReal() * ZOOM_MAGNIFICATION );

        }


      gl.glEnd();


    }
    /* draw lines */
    for (i=0; i<numberOfEdges; i++) {
      line = paper.getLine(i);

        gl.glLineWidth(LINEWIDTH);


      gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);

      gl.glBegin(GL.GL_LINES);

        gl.glVertex3f((float) line.getStartPoint().getXInReal() *
            ZOOM_MAGNIFICATION,
            (float) line.getStartPoint().getYInReal() * ZOOM_MAGNIFICATION,
            (float) line.getStartPoint().getZInReal() * ZOOM_MAGNIFICATION)
                ;

        gl.glVertex3f((float) line.getEndPoint().getXInReal() *
            ZOOM_MAGNIFICATION,
            (float) line.getEndPoint().getYInReal() * ZOOM_MAGNIFICATION,
            (float) line.getEndPoint().getZInReal() * ZOOM_MAGNIFICATION);

      gl.glEnd();
    }
  }

    // Methods required for the implementation of MouseListener
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {
      prevMouseX = e.getX();
      prevMouseY = e.getY();
      if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
        mouseRButtonDown = true;
      }

      if (mouseRButtonDown == true) {
        isAnimating = true;
      }


  }

  public void mouseReleased(MouseEvent e) {
    if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
      currentIndex_PaperGlId++;
      currentIndex_PaperGlId = currentIndex_PaperGlId % numberOfPapers;
      System.out.format("%d, %s\n", currentIndex_PaperGlId + 1, Integer.
          toString(currentIndex_PaperGlId + 1 ,2));
      mouseRButtonDown = false;
      }
  }

  public void mouseClicked(MouseEvent e) {}

  // Methods required for the implementation of MouseMotionListener

  public void mouseDragged(MouseEvent e) {
    if (mouseRButtonDown == false) {

      int x = e.getX();
      int y = e.getY();
      Dimension size = e.getComponent().getSize();

      float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
      float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);

      prevMouseX = x;
      prevMouseY = y;

      view_rotx += thetaX;
      view_roty += thetaY;
    }else {

    }
  }
  public void mouseMoved(MouseEvent e) {

  }

}
```

```java
package com.drancom.programmableMatter.folding.monitor;



import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.media.opengl.*;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.Point;
import com.drancom.programmableMatter.folding.controller.paper.Polygon;
import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
    ;
import com.drancom.programmableMatter.folding.origami.planner.
    PlannerForWiring;
import com.drancom.programmableMatter.folding.simulator.
    simulatorForOrigamis.PlanForOrigamis;
import com.sun.opengl.util.Animator;


public class MonitorOfPlanGroupOfPlanForOrigamis implements
    GLEventListener, MouseListener, MouseMotionListener {
  final static float LINEWIDTH = 2.0f;
  // final static float SIZE_OF_SQURE = 0.35f;
  final static float SIZE_OF_SQURE = 0.35f;

  // light
  // float pos0[] = {  -100.0f,  130.0f, 150.0f,  1.0f };
  float pos0[] = {  -100.0f,  100.0f, 100.0f,  1.0f };
  float ambientLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
  float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
  float specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};

  float white[] = {  1.0f,  1.0f,  1.0f,  0.2f };
  float red[] =   {  1.0f,  0.0f,  0.0f,  1.0f };
  float green[] = {  0.0f,  1.0f,  0.0f,  1.0f };
  float yellow[] = {  1.0f,  1.0f,  0.0f,  1.0f };
  float blue[] = { 0.0f, 0.0f, 1.0f, 0.0f };
  float black[] = { 0.0f, 0.0f, 0.0f, 1.0f };

  final static int SPEED_OF_ANIMATION = 15; //10 is default
  int counterForSpeedOfAnimation = 0;

  boolean isAnimating=false;


  PlanForOrigamis planForOrigamis;
  int    [][] planGroupTable;
  int    [][][] planTable;
  float  [][] edgeTable ;

  int numberOfGroups;



  public void run(PlanForOrigamis planForOrigamis) {
    numberOfGroups = planForOrigamis.getNumberOfGroups();

    paperGlId = new int[numberOfGroups];

    this.planForOrigamis = planForOrigamis;
    planTable = planForOrigamis.getPlanTable();
    planGroupTable = planForOrigamis.getPlanGroupTable();
    edgeTable = planForOrigamis.getEdgeTable();

    Frame frame = new Frame("Monitor Of PaperArray - Programmable Matter by
          Folding - by Byoungkwon An");
      GLCanvas canvas = new GLCanvas();

    canvas.addGLEventListener(this);
    frame.add(canvas);
    frame.setSize(800, 800);
    final Animator animator = new Animator(canvas);
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
            // Run this on another thread than the AWT event queue to
        // make sure the call to Animator.stop() completes before
        // exiting
        new Thread(new Runnable() {
          public void run() {
            animator.stop();
            System.exit(0);
          }
        }).start();
          }
    });
      frame.show();
      animator.start();
  }

  private float view_rotx = 0.0f, view_roty = 0.0f, view_rotz = 0.0f;

  private int paperGlId[];
  private int currentIndex_PaperGlId;

  private boolean isFolding = true;

  public boolean isFolding() {
    return isFolding;
  }

  private int prevMouseX, prevMouseY;
  private boolean mouseRButtonDown = false;

  public void init(GLAutoDrawable drawable) {
    int i;

    // Use debug pipeline
    // drawable.setGL(new DebugGL(drawable.getGL()));

    GL gl = drawable.getGL();

    System.err.println("INIT GL IS: " + gl.getClass().getName());
    System.err.println("Chosen GLCapabilities: " + drawable.getChosenGLCapabilities());

    gl.setSwapInterval(1);

    // Blend
    gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE);
    gl.glEnable(GL.GL_BLEND);


//      gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
//        gl.glClearDepth(1.0f);

    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos0, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambientLight, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuseLight, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular, 0);

    gl.glEnable(GL.GL_CULL_FACE);

    gl.glEnable(GL.GL_LIGHTING);
      gl.glEnable(GL.GL_LIGHT0);

//      gl.glEnable(GL.GL_DEPTH_TEST);

      /* make the papers */
      for (i=0; i < numberOfGroups; i++){
        paperGlId[i] = gl.glGenLists(1);
        gl.glNewList(paperGlId[i], GL.GL_COMPILE);
        buildGlPaper(gl, i, planForOrigamis);
        gl.glEndList();
      }

    currentIndex_PaperGlId = 0;


  //  gl.glEnable(GL.GL_NORMALIZE);

      drawable.addMouseListener(this);
      drawable.addMouseMotionListener(this);
  }




  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
    GL gl = drawable.getGL();

    float h = (float)height / (float)width;

    gl.glMatrixMode(GL.GL_PROJECTION);

    System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
    System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
    System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
    gl.glLoadIdentity();
    gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0.0f, 0.0f, -40.0f);
  }

  public void display(GLAutoDrawable drawable) {
    // Turn the gears' teeth

    // Get the GL corresponding to the drawable we are animating
    GL gl = drawable.getGL();

    // Special handling for the case where the GLJPanel is translucent
    // and wants to be composited with other Java 2D content
    if ((drawable instanceof GLJPanel) &&
        !((GLJPanel) drawable).isOpaque() &&
        ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
      gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
    } else {
      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    }


    // Rotate the entire assembly of gears based on how the user
    // dragged the mouse around
    gl.glPushMatrix();
    gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
    gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
    gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);

    // Place the first gear and call its display list
    gl.glPushMatrix();
      gl.glTranslatef(-5.0f, -5.0f, 0.0f);
      gl.glCallList(paperGlId[((int) currentIndex_PaperGlId)]);
    gl.glPopMatrix();


    // Remember that every push needs a pop; this one is paired with
    // rotating the entire gear assembly
    gl.glPopMatrix();

  }

  public void displayChanged(GLAutoDrawable drawable, boolean modeChanged, boolean deviceChanged) {}

  private void buildGlPaper(GL gl, int groupNumber, PlanForOrigamis planForOrigamis) {

    // for phase 0 and 1
    int i;
    int j;
    int k;
    int numberOfEdges = planForOrigamis.getNumberOfEdges();

    float [][] edgeTable = planForOrigamis.getEdgeTable();
    int   [][] planGroupTable = planForOrigamis.getPlanGroupTable();
    float [][][][] vertexs = planForOrigamis.getVertexs();

    gl.glShadeModel(GL.GL_FLAT);
    gl.glNormal3f(0.0f, 0.0f, 1.0f);

    /* draw lines */
    for (i=0; i<numberOfEdges; i++) {

        gl.glLineWidth(LINEWIDTH);

      if(planGroupTable[groupNumber][i] == 1) {
        // mount
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, red, 0);
      }else if (planGroupTable[groupNumber][i] == -1){
        // valley
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, blue, 0);

      }else {
        // just grid
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);
      }


      gl.glBegin(GL.GL_LINES);

      gl.glVertex3f((float)edgeTable[i][0]*8,
          (float)edgeTable[i][1]*8,
          (float)0.0*8);

      gl.glVertex3f((float)edgeTable[i][2]*8,
          (float)edgeTable[i][3]*8,
          (float)0.0*8);
      gl.glEnd();
    }

/**/

    for (i=0; i<9; i++) {
      for (j=0; j<9; j++){
        if(vertexs[groupNumber][i][j][3] == 1){
          gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, red, 0);

        } else
        if(vertexs[groupNumber][i][j][3] == 2){
          gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);


        }
        if(vertexs[groupNumber][i][j][3] == 1 || vertexs[groupNumber][i][j
             ][3] == 2)
        {
          gl.glBegin(GL.GL_LINES);
          gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 + (
               SIZE_OF_SQURE / 2),
              (float)vertexs[groupNumber][i][j][1]*8 + (SIZE_OF_SQURE / 2),
              (float)0.0*8);
          gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 + (
               SIZE_OF_SQURE / 2),
              (float)vertexs[groupNumber][i][j][1]*8 - (SIZE_OF_SQURE / 2),
              (float)0.0*8);

          gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 + (
               SIZE_OF_SQURE / 2),
              (float)vertexs[groupNumber][i][j][1]*8 - (SIZE_OF_SQURE / 2),
              (float)0.0*8);
          gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 - (
               SIZE_OF_SQURE / 2),
              (float)vertexs[groupNumber][i][j][1]*8 - (SIZE_OF_SQURE / 2),
              (float)0.0*8);

          gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 - (
               SIZE_OF_SQURE / 2),
              (float)vertexs[groupNumber][i][j][1]*8 - (SIZE_OF_SQURE / 2),
              (float)0.0*8);
          gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 - (
               SIZE_OF_SQURE / 2),
              (float)vertexs[groupNumber][i][j][1]*8 + (SIZE_OF_SQURE / 2),
              (float)0.0*8);

          gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 - (
               SIZE_OF_SQURE / 2),
              (float)vertexs[groupNumber][i][j][1]*8 + (SIZE_OF_SQURE / 2),
              (float)0.0*8);
          gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 + (
               SIZE_OF_SQURE / 2),
              (float)vertexs[groupNumber][i][j][1]*8 + (SIZE_OF_SQURE / 2),
              (float)0.0*8);

          gl.glEnd();
        }
      }

    }
/**/
  }

    // Methods required for the implementation of MouseListener
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {
      prevMouseX = e.getX();
      prevMouseY = e.getY();
      if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
        mouseRButtonDown = true;
      }

      if (mouseRButtonDown == true) {
        isAnimating = true;
      }


  }

  public void mouseReleased(MouseEvent e) {
    if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
      currentIndex_PaperGlId++;
      currentIndex_PaperGlId = currentIndex_PaperGlId % numberOfGroups;
      System.out.format("%d, %s\n", currentIndex_PaperGlId + 1, Integer.
           toString(currentIndex_PaperGlId + 1 ,2));
      mouseRButtonDown = false;
      }
  }

  public void mouseClicked(MouseEvent e) {}

  // Methods required for the implementation of MouseMotionListener
  public void mouseDragged(MouseEvent e) {
    if (mouseRButtonDown == false) {

      int x = e.getX();
      int y = e.getY();
      Dimension size = e.getComponent().getSize();

      float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
      float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);

      prevMouseX = x;
      prevMouseY = y;

      view_rotx += thetaX;
      view_roty += thetaY;
    }else {

    }
  }

  public void mouseMoved(MouseEvent e) {

  }

}
```

```
package com.drancom.programmableMatter.folding.monitor;

import java.awt.*;
import java.awt.event.*;

import javax.media.opengl.*;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.sun.opengl.util.*;

public class MonitorOfTileProgrammableMatter implements GLEventListener,
      MouseListener, MouseMotionListener {
  Paper[] papers;

  public void run(Paper[] papers) {
    this.papers = papers;
    paperGlId = new int[papers.length];

    Frame frame = new Frame("Programmable Matter by Folding");
      GLCanvas canvas = new GLCanvas();

      canvas.addGLEventListener(this);
      frame.add(canvas);
      frame.setSize(800, 800);
      final Animator animator = new Animator(canvas);
      frame.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {
            // Run this on another thread than the AWT event queue to
            // make sure the call to Animator.stop() completes before
            // exiting
            new Thread(new Runnable() {
                public void run() {
                  animator.stop();
                  System.exit(0);
                }
              }).start();
          }
        });
      frame.show();
      animator.start();
  }

  private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;

  private int paperGlId[];
  private int currentIndex_PaperGlId;

  private boolean isFolding = true;

  public boolean isFolding() {
    return isFolding;
  }
  private int prevMouseX, prevMouseY;
  private boolean mouseRButtonDown = false;

  public void init(GLAutoDrawable drawable) {
    int i;

    // Use debug pipeline
    // drawable.setGL(new DebugGL(drawable.getGL()));

    GL gl = drawable.getGL();

    System.err.println("INIT GL IS: " + gl.getClass().getName());

    System.err.println("Chosen GLCapabilities: " + drawable.
         getChosenGLCapabilities());

    gl.setSwapInterval(1);

    float pos[] = { 1.0f, 1.0f, -1.0f, 0.0f };
    // float pos[] = { 0.0f, 0.0f, -1.0f, 0.0f };
    float white[] = {1.0f, 1.0f, 1.0f, 1.0f};
    float red[] = { 0.8f, 0.1f, 0.0f, 1.0f };
    float green[] = { 0.0f, 0.8f, 0.2f, 1.0f };
    float blue[] = { 0.2f, 0.2f, 1.0f, 1.0f };
    float block[] = { 0.2f, 0.2f, 0.2f, 1.0f };

    gl.glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos, 0);
    gl.glEnable(GL.GL_CULL_FACE);
      gl.glEnable(GL.GL_LIGHTING);
      gl.glEnable(GL.GL_LIGHT0);
      gl.glEnable(GL.GL_DEPTH_TEST);

      gl.glLineWidth(0.5f);

      /* make the papers */
      for (i=0; i<papers.length; i++){
        paperGlId[i] = gl.glGenLists(1);
        gl.glNewList(paperGlId[i], GL.GL_COMPILE);
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT_AND_DIFFUSE,
             block, 0);
        buildGlPaper(gl, papers[i]);
        gl.glEndList();
      }

      currentIndex_PaperGlId = 0;



      gl.glEnable(GL.GL_NORMALIZE);

      drawable.addMouseListener(this);
      drawable.addMouseMotionListener(this);
  }

  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
        height) {
    GL gl = drawable.getGL();

    float h = (float)height / (float)width;

    gl.glMatrixMode(GL.GL_PROJECTION);

    System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
    System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
    System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
    gl.glLoadIdentity();
    gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0.0f, 0.0f, -40.0f);
  }

  public void display(GLAutoDrawable drawable) {
    // Turn the gears' teeth

    // Get the GL corresponding to the drawable we are animating
    GL gl = drawable.getGL();

    // Special handling for the case where the GLJPanel is translucent
    // and wants to be composited with other Java 2D content
    if ((drawable instanceof GLJPanel) &&
        !((GLJPanel) drawable).isOpaque() &&
        ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
      gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
    } else {
      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    }


    // Rotate the entire assembly of gears based on how the user
    // dragged the mouse around
    gl.glPushMatrix();
    gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
    gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
    gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);

    // Place the first gear and call its display list
    gl.glPushMatrix();
      gl.glTranslatef(-5.0f, -5.0f, 0.0f);
      gl.glCallList(paperGlId[currentIndex_PaperGlId]);
    gl.glPopMatrix();


    // Remember that every push needs a pop; this one is paired with
    // rotating the entire gear assembly
    gl.glPopMatrix();
    if (mouseRButtonDown == true){
      if (isFolding() == true ){
        currentIndex_PaperGlId--;

        if (currentIndex_PaperGlId < 0) {
          currentIndex_PaperGlId = 0;
        }

      } else {
        currentIndex_PaperGlId++;

        if (currentIndex_PaperGlId >= papers.length - 1 ) {
          currentIndex_PaperGlId = papers.length - 1;
        }
      }
    }
  }

  public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
       boolean deviceChanged) {}

  public static void buildGlPaper(GL gl, Paper paper){
    int i;
    int numberOfLine = paper.getNumberOfEdges();
    Vector startPointVector;
    Vector endPointVector ;

    gl.glShadeModel(GL.GL_FLAT);

    gl.glNormal3f(0.0f, 0.0f, 1.0f);

    /* draw lines */

    gl.glBegin(GL.GL_LINES);

    for (i=0; i<numberOfLine; i++) {

      startPointVector = paper.getLine(i).getStartPoint().getVectorInReal()
          ;
      endPointVector = paper.getLine(i).getEndPoint().getVectorInReal();

      gl.glVertex3f((float)startPointVector.getX()*8,
          (float)startPointVector.getY()*8,
          (float)startPointVector.getZ()*8);

      gl.glVertex3f((float)endPointVector.getX()*8,
          (float)endPointVector.getY()*8,
          (float)endPointVector.getZ()*8);
    }
    gl.glEnd();
  }

   // Methods required for the implementation of MouseListener
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {
      prevMouseX = e.getX();
      prevMouseY = e.getY();
      if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
        mouseRButtonDown = true;
      }
      if (mouseRButtonDown == true) {

        if (currentIndex_PaperGlId <= 0) {
          currentIndex_PaperGlId = 0;
          isFolding = false;
        } else if (currentIndex_PaperGlId >= papers.length - 1 ) {

           currentIndex_PaperGlId = papers.length - 1;
           isFolding = true;
      }

    }
  }

  public void mouseReleased(MouseEvent e) {
    if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
      mouseRButtonDown = false;
      }

  }

  public void mouseClicked(MouseEvent e) {}

  // Methods required for the implementation of MouseMotionListener
  public void mouseDragged(MouseEvent e) {
    if (mouseRButtonDown == false) {
      int x = e.getX();
      int y = e.getY();
      Dimension size = e.getComponent().getSize();

      float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
      float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);

      prevMouseX = x;
      prevMouseY = y;

      view_rotx += thetaX;
      view_roty += thetaY;
    }else {

    }
  }

  public void mouseMoved(MouseEvent e) {

  }
}
```

```
package com.drancom.programmableMatter.folding.monitor;

import java.awt.*;
import java.awt.event.*;

import javax.media.opengl.*;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.sun.opengl.util.*;
```

```

public class PlayWindow implements GLEventListener, MouseListener, MouseMotionListener {
  Paper[] papers;

  public void run(Paper[] papers) {
    this.papers = papers;
    paperGlId = new int[papers.length];

    Frame frame = new Frame("Programmable Matter by Folding");
      GLCanvas canvas = new GLCanvas();

      canvas.addGLEventListener(this);
      frame.add(canvas);
      frame.setSize(800, 800);
      final Animator animator = new Animator(canvas);
      frame.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {
            // Run this on another thread than the AWT event queue to
            // make sure the call to Animator.stop() completes before
            // exiting
            new Thread(new Runnable() {
                public void run() {
                  animator.stop();
                  System.exit(0);
                }
              }).start();
          }
        });
      frame.show();
      animator.start();
  }

  private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;

  private int paperGlId[];
  private int currentIndex_PaperGlId;

  private boolean isFolding = true;

  public boolean isFolding() {
    return isFolding;
  }
  private int prevMouseX, prevMouseY;
  private boolean mouseRButtonDown = false;

  public void init(GLAutoDrawable drawable) {
    int i;

    // Use debug pipeline
    // drawable.setGL(new DebugGL(drawable.getGL()));

    GL gl = drawable.getGL();

    System.err.println("INIT GL IS: " + gl.getClass().getName());

    System.err.println("Chosen GLCapabilities: " + drawable.getChosenGLCapabilities());

    gl.setSwapInterval(1);

    float pos[] = { 1.0f, 1.0f, -1.0f, 0.0f };
    // float pos[] = { 0.0f, 0.0f, -1.0f, 0.0f };
    float white[] = {1.0f, 1.0f, 1.0f, 1.0f};
    float red[] = { 0.8f, 0.1f, 0.0f, 1.0f };
    float green[] = { 0.0f, 0.8f, 0.2f, 1.0f };
    float blue[] = { 0.2f, 0.2f, 1.0f, 1.0f };
    float block[] = { 0.2f, 0.2f, 0.2f, 1.0f };

    gl.glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos, 0);
    gl.glEnable(GL.GL_CULL_FACE);
      gl.glEnable(GL.GL_LIGHTING);
      gl.glEnable(GL.GL_LIGHT0);
      gl.glEnable(GL.GL_DEPTH_TEST);

      gl.glLineWidth(0.5f);

      /* make the papers */
      for (i=0; i<papers.length; i++){
        paperGlId[i] = gl.glGenLists(1);
        gl.glNewList(paperGlId[i], GL.GL_COMPILE);
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT_AND_DIFFUSE, block, 0);
        buildGlPaper(gl, papers[i]);
        gl.glEndList();
      }

      currentIndex_PaperGlId = 0;



      gl.glEnable(GL.GL_NORMALIZE);

      drawable.addMouseListener(this);
      drawable.addMouseMotionListener(this);
  }

  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
    GL gl = drawable.getGL();

    float h = (float)height / (float)width;

    gl.glMatrixMode(GL.GL_PROJECTION);

    System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
    System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
    System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
    gl.glLoadIdentity();
    gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0.0f, 0.0f, -40.0f);
  }

  public void display(GLAutoDrawable drawable) {
    // Turn the gears' teeth

    // Get the GL corresponding to the drawable we are animating
    GL gl = drawable.getGL();

    // Special handling for the case where the GLJPanel is translucent
    // and wants to be composited with other Java 2D content
    if ((drawable instanceof GLJPanel) &&
        !((GLJPanel) drawable).isOpaque() &&
        ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
      gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
    } else {
      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    }


    // Rotate the entire assembly of gears based on how the user
    // dragged the mouse around
    gl.glPushMatrix();
    gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
    gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
    gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);

    // Place the first gear and call its display list
    gl.glPushMatrix();
      gl.glTranslatef(-5.0f, -5.0f, 0.0f);
      gl.glCallList(paperGlId[currentIndex_PaperGlId]);
    gl.glPopMatrix();


    // Remember that every push needs a pop; this one is paired with
    // rotating the entire gear assembly
    gl.glPopMatrix();
    if (mouseRButtonDown == true){
      if (isFolding() == true ){
        currentIndex_PaperGlId--;

        if (currentIndex_PaperGlId < 0) {
          currentIndex_PaperGlId = 0;
        }

      } else {
        currentIndex_PaperGlId++;

        if (currentIndex_PaperGlId >= papers.length - 1 ) {
          currentIndex_PaperGlId = papers.length - 1;
        }
      }
    }
  }

  public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
       boolean deviceChanged) {}

  public static void buildGlPaper(GL gl, Paper paper){
    int i;
    int numberOfLine = paper.getNumberOfEdges();
    Vector startPointVector;
    Vector endPointVector ;

    gl.glShadeModel(GL.GL_FLAT);

    gl.glNormal3f(0.0f, 0.0f, 1.0f);

    /* draw lines */

    gl.glBegin(GL.GL_LINES);

    for (i=0; i<numberOfLine; i++) {

      startPointVector = paper.getLine(i).getStartPoint().getVectorInReal()
           ;
      endPointVector = paper.getLine(i).getEndPoint().getVectorInReal();

      gl.glVertex3f((float)startPointVector.getX()*8,
          (float)startPointVector.getY()*8,
          (float)startPointVector.getZ()*8);

      gl.glVertex3f((float)endPointVector.getX()*8,
          (float)endPointVector.getY()*8,
          (float)endPointVector.getZ()*8);
    }
    gl.glEnd();
  }

    // Methods required for the implementation of MouseListener
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {
      prevMouseX = e.getX();
      prevMouseY = e.getY();
      if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
        mouseRButtonDown = true;
      }
      if (mouseRButtonDown == true) {

        if (currentIndex_PaperGlId <= 0) {
          currentIndex_PaperGlId = 0;
          isFolding = false;
        } else if (currentIndex_PaperGlId >= papers.length - 1 ) {

           currentIndex_PaperGlId = papers.length - 1;
           isFolding = true;
      }

    }
  }

  public void mouseReleased(MouseEvent e) {
    if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
      mouseRButtonDown = false;
      }

  }

  public void mouseClicked(MouseEvent e) {}

  // Methods required for the implementation of MouseMotionListener
  public void mouseDragged(MouseEvent e) {
    if (mouseRButtonDown == false) {
      int x = e.getX();
      int y = e.getY();
      Dimension size = e.getComponent().getSize();

      float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
      float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);

      prevMouseX = x;
      prevMouseY = y;

      view_rotx += thetaX;
      view_roty += thetaY;
    }else {

    }
  }

  public void mouseMoved(MouseEvent e) {

  }
}
```

```
package com.drancom.programmableMatter.folding.monitor;

import java.awt.*;
import java.awt.event.*;

import javax.media.opengl.*;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.
    UnfoldingPaper;
import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.sun.opengl.util.*;

public class UnfoldingWindow extends MainWindow implements GLEventListener,
      MouseListener, MouseMotionListener {
  Paper paper;

  GLCanvas canvas;

  final static float pos[] = { 1.0f, 1.0f, -1.0f, 0.0f };
  // float pos[] = { 0.0f, 0.0f, -1.0f, 0.0f };
  final static float white[] = {1.0f, 1.0f, 1.0f, 1.0f};
  final static float red[] = { 0.8f, 0.1f, 0.0f, 1.0f };
  final static float green[] = { 0.0f, 0.8f, 0.2f, 1.0f };
  final static float blue[] = { 0.2f, 0.2f, 1.0f, 1.0f };
  final static float block[] = { 0.2f, 0.2f, 0.2f, 1.0f };


  private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;

  private int paperGlId;
  private int currentIndex_PaperGlId;

  private boolean isFolding = true;


  private int prevMouseX, prevMouseY;
  private boolean mouseRButtonDown = false;

  public void run(Paper paper) {
    this.paper = paper;

    Frame frame = new Frame("Programmable Matter by Folding");
      canvas = new GLCanvas();

      canvas.addGLEventListener(this);
      frame.add(canvas);
      frame.setSize(800, 800);
      final Animator animator = new Animator(canvas);
      frame.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {
            // Run this on another thread than the AWT event queue to
            // make sure the call to Animator.stop() completes before
            // exiting
            new Thread(new Runnable() {
                public void run() {
                  animator.stop();
                  System.exit(0);
                }
              }).start();
          }
        });
      frame.show();
      animator.start();
  }

  public void init(GLAutoDrawable drawable) {
    int i;

    // Use debug pipeline
    // drawable.setGL(new DebugGL(drawable.getGL()));

    GL gl = drawable.getGL();

    System.err.println("INIT GL IS: " + gl.getClass().getName());

    System.err.println("Chosen GLCapabilities: " + drawable.
         getChosenGLCapabilities());

    gl.setSwapInterval(1);


    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos, 0);
    gl.glEnable(GL.GL_CULL_FACE);
      gl.glEnable(GL.GL_LIGHTING);
      gl.glEnable(GL.GL_LIGHT0);
      gl.glEnable(GL.GL_DEPTH_TEST);

      gl.glLineWidth(0.5f);
  /*
      for (i=0; i<papers.length; i++){
        paperGlId[i] = gl.glGenLists(1);
        gl.glNewList(paperGlId[i], GL.GL_COMPILE);
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT_AND_DIFFUSE,
             white, 0);
        buildGlPaper(gl, papers[i]);
        gl.glEndList();
      }
*/
      currentIndex_PaperGlId = 0;

      gl.glEnable(GL.GL_NORMALIZE);

      drawable.addMouseListener(this);
      drawable.addMouseMotionListener(this);
  }

  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
        height) {
    GL gl = drawable.getGL();

    float h = (float)height / (float)width;

    gl.glMatrixMode(GL.GL_PROJECTION);

    System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
    System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
    System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
    gl.glLoadIdentity();
    gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0.0f, 0.0f, -40.0f);
  }

  public void display(GLAutoDrawable drawable) {
    // Turn the gears' teeth

    // Get the GL corresponding to the drawable we are animating
    GL gl = drawable.getGL();

    // Special handling for the case where the GLJPanel is translucent
    // and wants to be composited with other Java 2D content
    if ((drawable instanceof GLJPanel) &&
        !((GLJPanel) drawable).isOpaque() &&
        ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
      gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
    } else {
      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    }


    // Rotate the entire assembly of gears based on how the user
    // dragged the mouse around
    gl.glPushMatrix();
    gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
    gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
    gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);

    // Place the first gear and call its display list
    gl.glPushMatrix();
      gl.glTranslatef(-5.0f, -5.0f, 0.0f);
      //gl.glCallList(paperGlId[currentIndex_PaperGlId]);
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT_AND_DIFFUSE,
             white, 0);
        buildGlPaper(gl, paper);

    gl.glPopMatrix();


    // Remember that every push needs a pop; this one is paired with
    // rotating the entire gear assembly
    gl.glPopMatrix();
  }

  public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
       boolean deviceChanged) {}

  public boolean isFolding() {
    return isFolding;
  }

  public void buildGlPaper(GL gl, Paper paper){
    int i;
    int numberOfLine = paper.getNumberOfEdges();
    Vector startPointVector;
    Vector endPointVector ;

    gl.glShadeModel(GL.GL_FLAT);

    gl.glNormal3f(0.0f, 0.0f, 1.0f);

    /* draw lines */

    gl.glBegin(GL.GL_LINES);

    for (i=0; i<numberOfLine; i++) {

      startPointVector = paper.getLine(i).getStartPoint().getVectorInReal()
          ;
      endPointVector = paper.getLine(i).getEndPoint().getVectorInReal();

      gl.glVertex3f((float)startPointVector.getX()*8,
          (float)startPointVector.getY()*8,
          (float)startPointVector.getZ()*8);

      gl.glVertex3f((float)endPointVector.getX()*8,
          (float)endPointVector.getY()*8,
          (float)endPointVector.getZ()*8);
    }
    gl.glEnd();
  }

    // Methods required for the implementation of MouseListener
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {
      prevMouseX = e.getX();
      prevMouseY = e.getY();
      if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
        mouseRButtonDown = true;
      }
      if (mouseRButtonDown == true) {
//            animator.stop();

          final Animator animator = new Animator(canvas);

        animator.stop();
        Paper snapshotPaper = paper.snapshot();

//          ((UnfoldingPaper) snapshotPaper).unfolding(0.1f);

        paper = snapshotPaper;
          animator.start();

    }
  }

  public void mouseReleased(MouseEvent e) {
    if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
      mouseRButtonDown = false;

      }

  }

  public void mouseClicked(MouseEvent e) {}

  // Methods required for the implementation of MouseMotionListener
  public void mouseDragged(MouseEvent e) {
    if (mouseRButtonDown == false) {
      int x = e.getX();
      int y = e.getY();
      Dimension size = e.getComponent().getSize();

      float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
      float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);

      prevMouseX = x;
      prevMouseY = y;

      view_rotx += thetaX;
      view_roty += thetaY;
    }else {

    }
  }

  public void mouseMoved(MouseEvent e) {

  }
}
```

```
package com.drancom.programmableMatter.folding.origami.planner;

public interface Plan {

}
```

```
package com.drancom.programmableMatter.folding.origami.planner;

public class PlanForAngleActurator implements Plan {

}
```

```
package com.drancom.programmableMatter.folding.origami.planner;

public class PlanForWiring implements Plan {
  // folding or unfolding []
  // phases []
  // inside or outside[]
  // numberOfEdge []
  // data true or false
  boolean planTable[][][][];
  int numberOfPhases;
  int numberOfEdges;

  PlanForWiring (int maxPhases, int numberOfEdges) {
    int i;
    int j;
    int k;
    int l;

    numberOfPhases = 0;
    this.numberOfEdges = numberOfEdges;
    planTable = new boolean[2][maxPhases][2][numberOfEdges];

    for (i=0; i<2; i++) {
      for (j=0; j<maxPhases; j++) {
        for(k=0; k<2 ; k++) {
          for(l=0; l<numberOfEdges; l++) {
            planTable[i][j][k][l] = false;
          }
        }
      }
    }
  }

  public boolean setPlanForWiring(int folding
      , int phase
      , int inside
      , int edgeNumber
      , boolean active) {

    planTable[folding][phase][inside][edgeNumber] = active;

    return true;
  }

  public void setNumberOfPhases(int numberOfPhases) {
    this.numberOfPhases = numberOfPhases;
  }

  public boolean [][][][] getPlanTable() {
    return planTable;
  }

  public int getNumberOfPhases() {
    return numberOfPhases;
  }

  public int getNumberOfEdges() {
    return numberOfEdges;
  }
}
```

```
package com.drancom.programmableMatter.folding.origami.planner;

import com.drancom.programmableMatter.folding.controller.paper.Paper;

public interface Planner {

  public void build(Paper[] papers);
  public void build(Plan[] plans);

  public Plan getPlan();
  public void exportPlan(String fileName);
  public void exportPlan(String fileName, Paper[] papers);
}
```

```
package com.drancom.programmableMatter.folding.origami.planner;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FileAngleData;
import com.drancom.programmableMatter.folding.dataFile.FilePlan;
import com.drancom.programmableMatter.folding.dataFile.
      FilePlanForAngleActuration;

public class PlannerForAngleActurator implements Planner {

  public void build(Paper[] papers) {

    int i;
    int j;

    int numberOfLines;
    float levelOfActuratingPower;

    numberOfLines = papers[0].getNumberOfEdges();

    for (i=0; i<papers.length -1 ; i++) {
      for(j=0 ; j<numberOfLines ; j++) {
        levelOfActuratingPower = papers[i].getLine(j).getAngle() - papers[i
            +1].getLine(j).getAngle();

        if (levelOfActuratingPower > 0.15f) {
          papers[i+1].getLine(j).setLevelOfActuratingPower( 2.0f);
        } else if (levelOfActuratingPower < -0.15) {
          papers[i+1].getLine(j).setLevelOfActuratingPower(-2.0f);
        } else if (levelOfActuratingPower > 0.0) {
          papers[i+1].getLine(j).setLevelOfActuratingPower( 1.0f);
        } else if (levelOfActuratingPower < 0.0) {
          papers[i+1].getLine(j).setLevelOfActuratingPower(-1.0f);
        } else {
          papers[i+1].getLine(j).setLevelOfActuratingPower(0.0f);
        }
      }
    }

    for(j=0 ; j<numberOfLines ; j++) {

      papers[0].getLine(j).setLevelOfActuratingPower(0.0f);

    }
  }

  public void exportPlan(String fileName, Paper[] papers) {
    FilePlan filePlan = new FilePlanForAngleActuration();

    filePlan.build(fileName, papers);
  }

  @Override
  public Plan getPlan() {
    return null;
  }

  @Override
  public void build(Plan[] plans) {
    // TODO Auto-generated method stub

  }

  @Override
  public void exportPlan(String fileName) {
    // TODO Auto-generated method stub

  }

}
```

```
package com.drancom.programmableMatter.folding.origami.planner;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FilePlan;
import com.drancom.programmableMatter.folding.dataFile.FilePlanForWiring;

```

```java

public class PlannerForWiring implements Planner {

  final int MAX_PHEASES = 10;
  final float STANDARD_NUMBER_FOR_ACTIVATION = 0.0f;
  final int STANDARD_NUMBER_OF_PHASE_COUNTER_FOR_PHASE_CHANGE = 4;


  PlanForWiring planForWiring;

  public void build(Paper[] papers) {

    int i;
    int j;

    int numberOfLines;
    float levelOfActuratingPower;

    int phase = 0;
    int phaseCounter = 0;

    float standard_number_for_Activation = STANDARD_NUMBER_FOR_ACTIVATION;

    numberOfLines = papers[0].getNumberOfEdges();
    planForWiring = new PlanForWiring(10, numberOfLines);



    for (i=0; i<papers.length -1 ; i++) {
      for(j=0 ; j<numberOfLines ; j++) {
        levelOfActuratingPower = papers[i].getLine(j).getAngle() - papers[i
             +1].getLine(j).getAngle();

        if (levelOfActuratingPower > standard_number_for_Activation) {
          if (phaseCounter >=
               STANDARD_NUMBER_OF_PHASE_COUNTER_FOR_PHASE_CHANGE *
               numberOfLines) {
            phase++;
            planForWiring.setNumberOfPhases(phase);
          }

          planForWiring.setPlanForWiring(0, phase, 0, j, true);
                phaseCounter = 0;

        } else if (levelOfActuratingPower < -1 *
             standard_number_for_Activation ) {
          if (phaseCounter >=
               STANDARD_NUMBER_OF_PHASE_COUNTER_FOR_PHASE_CHANGE *
               numberOfLines) {
            phase++;
            planForWiring.setNumberOfPhases(phase);
          }

          planForWiring.setPlanForWiring(0, phase, 1, j, true);
                phaseCounter = 0;
        } else {
          phaseCounter++;
        }
      }
    }
  }

  public void exportPlan(String fileName, Paper[] papers) {
    FilePlan filePlan = new FilePlanForWiring();

    filePlan.build(fileName, papers);
  }

  public Plan getPlan() {

    return (Plan) planForWiring;
  }

  @Override
  public void build(Plan[] plans) {
    // TODO Auto-generated method stub

  }

  @Override
  public void exportPlan(String fileName) {
    // TODO Auto-generated method stub

  }
}
```

```java
package com.drancom.programmableMatter.folding.simulator;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FileAngleData;
import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.MainWindow;

public class AngleDataCollector {

/** /
    public static final String FILENAME = "c:\\foldingdata\\save_airplain\\m
          %05d.obj";
    public static final String FILEANGLENAME = "c:\\foldingdata\\
          save_airplain\\angle.csv";
    public static final int NUMBER_OF_FILES= 50;
/** /
  public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d.
        obj";
  public static final String FILEANGLENAME = "c:\\foldingdata\\save_box\\
        angle.csv";
  public static final int NUMBER_OF_FILES= 70;
/** /
  public static final String FILENAME = "c:\\foldingdata\\save_cup\\m%05d.
        obj";
  public static final String FILEANGLENAME = "c:\\foldingdata\\save_cup\\
        angle.csv";
  public static final int NUMBER_OF_FILES= 140;
/**/
  public static final String PLAN_FILENAME = "c:\\foldingdata\\save_bench\\
        plan_save_bench.csv";
  public static final String FILENAME = "c:\\foldingdata\\save_bench\\m%05d
        .obj";
  public static final String FILEANGLENAME = "c:\\foldingdata\\save_bench\\
        angle.csv";
  public static final int NUMBER_OF_FILES= 70;
/**/
//  public static final String FILENAME = "c:\\foldingdata\\save_box\\
      m00070.obj";
//  public static final int NUMBER_OF_FILES= 1;

  void run() {

    int i;
    String fileName;
    String fileAngleName;

    Paper[] papers;
    FileObj[] fileObjs;
    FileAngleData fileAngleData;

    // init
    papers = new Paper[NUMBER_OF_FILES];
    fileObjs = new FileObj[NUMBER_OF_FILES];
    fileAngleData = new FileAngleData();

    // load
    for (i=0; i < NUMBER_OF_FILES; i++) {

      fileName = String.format(FILENAME, i+1);
      papers[i] = new Paper();
      fileObjs[i] = new FileObj();
      fileObjs[i].load(fileName, papers[i]);
    }

    // save
    fileAngleName = String.format(FILEANGLENAME);
    fileAngleData.build(fileAngleName, papers);
  }

  public static void main(String[] args) {
    // TODO Auto-generated method stub
    AngleDataCollector simulator = new AngleDataCollector();
    simulator.run();

  }
}
```

```
package com.drancom.programmableMatter.folding.simulator;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.MainWindow;

public class Player {
/**/

  public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
       RigidOrigami006\\RigidOrigami\\save_4x4_s-shuttle\\m%05d.obj";
  public static final int NUMBER_OF_FILES= 40;
/** /
  public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
       RigidOrigami006\\RigidOrigami\\save_4x4_pyramid\\m%05d.obj";
  public static final int NUMBER_OF_FILES= 40;
/** /
  public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
       RigidOrigami006\\RigidOrigami\\save_8x8_s-shuttle\\m%05d.obj";
  public static final int NUMBER_OF_FILES= 50;
/** /
  public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
       RigidOrigami006\\RigidOrigami\\save_8x8_hat\\m%05d.obj";
  public static final int NUMBER_OF_FILES= 28;
/** /
  public static final String FILENAME = "c:\\foldingdata\\save_airplain\\m
       %05d.obj";
  public static final int NUMBER_OF_FILES= 70;
/** /
  public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d.
       obj";
  public static final int NUMBER_OF_FILES= 70;
/**/
//  public static final String FILENAME = "c:\\foldingdata\\save_box\\
     m00070.obj";
//  public static final int NUMBER_OF_FILES= 1;

  Paper[] papers;
  FileObj[] fileObjs;
  MainWindow mainWindow;
  public Player() {

  }

  void run() {
    int i;
    String fileName;

    // init
    papers = new Paper[NUMBER_OF_FILES];
    fileObjs = new FileObj[NUMBER_OF_FILES];
    mainWindow = new MainWindow();

    // load
    for (i=0; i < NUMBER_OF_FILES; i++) {

      fileName = String.format(FILENAME, i+1);
      papers[i] = new Paper();
      fileObjs[i] = new FileObj();
      fileObjs[i].load(fileName, papers[i]);

    }

    mainWindow.run(papers);

  }


  public static void main(String[] args) {
    // TODO Auto-generated method stub
    Player simulator = new Player();
    simulator.run();
  }
}
```

```
package com.drancom.programmableMatter.folding.simulator;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.MainWindow;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.origami.planner.
     PlannerForWiring;

public class Simulator {

/** /
  public static final String PLAN_FILENAME = "c:\\foldingdata\\
       save_airplain\\plan_airplain.csv";
  public static final String FILENAME = "c:\\foldingdata\\save_airplain\\m
       %05d.obj";
  public static final int NUMBER_OF_FILES= 50;

/** /
  public static final String PLAN_FILENAME = "c:\\foldingdata\\save_box\\
       plan_box.csv";
  public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d.
       obj";
  public static final int NUMBER_OF_FILES= 70;

/** /
  public static final String PLAN_FILENAME = "c:\\foldingdata\\
       save_sailboat2\\plan_sailboat2.csv";
  public static final String FILENAME = "c:\\foldingdata\\save_sailboat2\\m
       %05d.obj";
  public static final int NUMBER_OF_FILES= 35;
/**/
  public static final String PLAN_FILENAME = "c:\\foldingdata\\
       save_8x8bench\\plan_save_bench.csv";
  public static final String FILENAME = "c:\\foldingdata\\save_8x8bench\\m
       %05d.obj";
  public static final int NUMBER_OF_FILES= 70;
/**/

// public static final String FILENAME = "c:\\foldingdata\\save_box\\
      m00070.obj";
// public static final int NUMBER_OF_FILES= 1;

  Paper[] papers;
  FileObj[] fileObjs;
  MainWindow mainWindow;
  public Simulator() {
  }

  void run() {
    int i;
    String fileName;

    // init
    papers = new Paper[NUMBER_OF_FILES];
    fileObjs = new FileObj[NUMBER_OF_FILES];
    mainWindow = new MainWindow();

    // load
    for (i=0; i < NUMBER_OF_FILES; i++) {
      fileName = String.format(FILENAME, i+1);
      papers[i] = new Paper();
      fileObjs[i] = new FileObj();
      fileObjs[i].load(fileName, papers[i]);
    }

    Planner planer = new PlannerForWiring();

    planer.build(papers);

    planer.exportPlan(PLAN_FILENAME, papers);

    mainWindow.run(papers);

  }

  public static void main(String[] args) {
    // TODO Auto-generated method stub
    Simulator simulator = new Simulator();
    simulator.run();
  }
}
```

```
package com.drancom.programmableMatter.folding.simulator;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.MainWindow;
import com.drancom.programmableMatter.folding.monitor.
      MainWindowForFoldingRobotWiring;
import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
      ;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.origami.planner.
      PlannerForWiring;

public class SimulatorForFoildingWithWire {

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_airplain\\plan_for_wiring_airplain.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_airplain\\
          m%05d.obj";
    public static final int NUMBER_OF_FILES= 50;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_box\\
          plan_for_wiring_box.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d
          .obj";
    public static final int NUMBER_OF_FILES= 70;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_sailboat2\\plan_for_wiring_sailboat2.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_sailboat2
          \\m%05d.obj";
    public static final int NUMBER_OF_FILES= 35;

  /**/
    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_bench
          \\plan_for_wiring_save_bench.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_bench\\m
          %05d.obj";
    public static final int NUMBER_OF_FILES= 70;

  /**/

//    public static final String FILENAME = "c:\\foldingdata\\save_box\\
      m00070.obj";
//    public static final int NUMBER_OF_FILES= 1;

    Paper[] papers;
    FileObj[] fileObjs;
    MainWindowForFoldingRobotWiring mainWindow;
    public SimulatorForFoildingWithWire() {

    }

    void run() {
      int i;
      String fileName;

      // Initiation
      papers = new Paper[NUMBER_OF_FILES];
      fileObjs = new FileObj[NUMBER_OF_FILES];
       mainWindow = new MainWindowForFoldingRobotWiring();

      // load
      for (i=0; i < NUMBER_OF_FILES; i++) {
        fileName = String.format(FILENAME, i+1);
        papers[i] = new Paper();
        fileObjs[i] = new FileObj();
        fileObjs[i].load(fileName, papers[i]);
      }

      Planner planer = new PlannerForWiring(); // new planer

      planer.build(papers);

      planer.exportPlan(PLAN_FILENAME, papers);

      mainWindow.run(papers, (PlanForWiring) planer.getPlan());

    }

    public static void main(String[] args) {
      // TODO Auto-generated method stub
      SimulatorForFoildingWithWire simulator = new
            SimulatorForFoildingWithWire ();
      simulator.run();
    }
}
```

```
package com.drancom.programmableMatter.folding.simulator;
```

```java

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.
      UnfoldingPaper;
import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.MainWindow;
import com.drancom.programmableMatter.folding.monitor.UnfoldingWindow;

public class UnfoldingSimulator {
/**/
  public static final String FILENAME = "c:\\foldingdata\\save_airplain\\m
        %05d.obj";
  public static final int NUMBER_OF_FILES= 81;
/** /
  public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d.
        obj";
  public static final int NUMBER_OF_FILES= 70;
/**/
//  public static final String FILENAME = "c:\\foldingdata\\save_box\\
      m00070.obj";
//  public static final int NUMBER_OF_FILES= 1;
/**/

  Paper paper;
  FileObj fileObj;
  MainWindow mainWindow;
  public UnfoldingSimulator() {
  }

  void run() {
    int i;
    String fileName;

    // init
    paper = new UnfoldingPaper();
    fileObj = new FileObj();
    mainWindow = new UnfoldingWindow();

    // load
    fileName = String.format(FILENAME, 1);
    fileObj.load(fileName, paper);

    ((UnfoldingWindow)mainWindow).run(paper);

  }


  public static void main(String[] args) {
    // TODO Auto-generated method stub
    UnfoldingSimulator simulator = new UnfoldingSimulator();
    simulator.run();
  }
}
```

```java
package com.drancom.programmableMatter.folding.simulator.boxcorner;

import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
      .MainWindowForFoldingRobotOrigami;

public class BoxCornerSimulator {
  final static double DEFULT_STEP = 0.02f;
  final static String FILENAME = "C:\\foldingdata\\boxCorner\\BoxCornerData
        .csv";

  public void run(){
    int i;

    double theta;
    double cos;
    double sin;

    double a;
    double b;
    double c;


    int numberOfLevels;

    numberOfLevels = ((int)((Math.PI) /2  / DEFULT_STEP)) + 1;

    double [] origin = new double[3];

    System.out.format("c'-d'-e\n");
    System.out.format("|'\\| /|\n");
    System.out.format("b'-o'-d\n");
    System.out.format("| /|'\\|\n");
    System.out.format("a -b -c\n");
    System.out.format("\n");

    System.out.format("7 -6 -5\n");
    System.out.format("|'\\| /|\n");
    System.out.format("8 -0 -4\n");
    System.out.format("| /| \\|\n");
    System.out.format("1 -2 -3\n");
    System.out.format("\n");
    System.out.format("\n");


    double [][][] points = new double [numberOfLevels][9][3];
    double [][] angles = new double [numberOfLevels][9];

    double [][][] pointOfMagnets = new double [numberOfLevels][2][3];
    Vector [][] unitVectorOfMagnets = new Vector [numberOfLevels][2];

    Vector v1, v2;

    double [] distancesBetweenMagnets = new double [numberOfLevels];

    theta=Math.PI + DEFULT_STEP;
    for (i = 0; i < numberOfLevels; i++){

      theta-=DEFULT_STEP;
      cos=Math.cos(Math.PI - theta);
      sin=Math.sin(Math.PI - theta);

      // 0 = {         0.0f,          0.0f,          0.0f}
      // 1 = {        -1.0f,         -1.0f,          0.0f}
      // 2 = {         0.0f,         -1.0f,          0.0f}
      // 3 = {COS(PI-Th2),           -1.0f,  SIN(PI-TH2)}
      // 4 = {COS(PI-Th2),            0.0f,  SIN(PI-TH2)}
      // 5 = { (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta]^3 -
//            (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta]^3 -
//            Cos[theta] Sin[theta]^2 + \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2]
      Sin[theta]^2 +
//              4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 +
//              4 Sin[theta]^4 + 4 Sqrt[2] Sin[theta]^4 -
//              4 Cos[theta]^2 Sin[theta]^4 - 2 Sin[theta]^6))/(2 (Cos[
//              theta]^2 + 2 Sin[theta]^2)),
//          (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta]^3 -
//              Cos[theta] Sin[
//              theta]^2 + \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2] Sin[theta]^2
      +
//              4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 +
//              4 Sin[theta]^4 + 4 Sqrt[2] Sin[theta]^4 -
//              4 Cos[theta]^2 Sin[theta]^4 - 2 Sin[theta]^6))/(2 (Cos[
     theta]^2 +
//            2 Sin[theta]^2)),
//          \[Sqrt](Sqrt[2] - (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta
        ]^[theta]
//            Cos[theta] Sin[1]^2 + \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2] Sin
        [theta]^2 +
//            4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 + 4
         Sin[theta]^4 +
//            4 Sqrt[2] Sin[theta]^4 - 4 Cos[theta]^2 Sin[theta]^4 -
//            2 Sin[theta]^6))^2/(2 (Cos[theta]^2 + 2 Sin[theta]^2)^2))

        // 6 = {        0.0,    COS(PI-Th2),    SIN(PI-TH2)}
        // 7 = {       -1.0,    COS(PI-Th2),    SIN(PI-TH2)}
        // 8 = {        0.0,          -1.0,             0.0}

        // 0 = {        0.0,           0.0,             0.0}
        points[i][0][0] =  0.0;
        points[i][0][1] =  0.0;
        points[i][0][2] =  0.0;

        // 1 = {       -1.0,          -1.0,             0.0}
        points[i][1][0] = -1.0;
        points[i][1][1] = -1.0;
        points[i][1][2] =  0.0;

        // 2 = {        0.0,          -1.0,             0.0}
        points[i][2][0] =  0.0;
        points[i][2][1] = -1.0;
        points[i][2][2] =  0.0;

        // 3 = {COS(PI - theta),        -1.0, SIN(PI-theta)}
        //       ==  -1 * COS(theta)               SIN(Theta)
        points[i][3][0] = cos;
        points[i][3][1] = -1.0;
        points[i][3][2] = sin;

        // 4 = {COS(PI-Th2),           0.0, SIN(PI-TH2)}
        //       ==  -1 * COS(theta)
        points[i][4][0] =  cos;
        points[i][4][1] =  0.0;
        points[i][4][2] =  sin;

        a = points[i][4][0];
        b = points[i][4][1];
        c = points[i][4][2];


        // 5
//        {{z -> (4 c - 4 a^2 c - 4 b^2 c -
//                  4 c^3 - \[Sqrt]((-4 c + 4 a^2 c + 4 b^2 c + 4 c^3)^2 -
//                     4 (-a^2 - 2 a b - b^2 - 2 c^2) (-2 + (4 + Sqrt[2]) a^2 -
//                        2 a^4 + 2 Sqrt[2] a b + (4 + Sqrt[2]) b^2 - 4 a^2 b^2
        -

//                        2 b^4 + 4 c^2 - 4 a^2 c^2 - 4 b^2 c^2 -
//                        2 c^4)))/(2 (-a^2 - 2 a b - b^2 - 2 c^2))},


//         {z -> (4 c -
//                  4 a^2 c - 4 b^2 c -
//                  4 c^3 + \[Sqrt]((-4 c + 4 a^2 c + 4 b^2 c + 4 c^3)^2 -
//                     4 (-a^2 - 2 a b - b^2 - 2 c^2) (-2 + (4 + Sqrt[2]) a^2 -
//                        2 a^4 + 2 Sqrt[2] a b + (4 + Sqrt[2]) b^2 - 4 a^2 b^2
        -

//                        2 b^4 + 4 c^2 - 4 a^2 c^2 - 4 b^2 c^2 -
//                        2 c^4)))/(2 (-a^2 - 2 a b - b^2 - 2 c^2))}}

/** /
        points[i][5][2] = (4 * c -
                4 * Math.pow(a,2) * c - 4 * Math.pow(b,2) * c -
                4 * Math.pow(c,3) + Math.sqrt(Math.pow((-4 * c + 4 * Math.pow
                    (a,2) * c + 4 * Math.pow(b,2) * c + 4 * Math.pow(c,3))
                    ,2) -
                4 * (-1 * Math.pow(a,2) - 2 * a * b - Math.pow(b,2) - 2 *
                    Math.pow(c,2) * (-2 + (4 + Math.sqrt(2)) * Math.pow(a
                    ,2) -
                  2 * Math.pow(a,4) + 2 * Math.sqrt(2) * a * b + (4 +
                      Math.sqrt(2))* Math.pow(b,2) - 4 * Math.pow(a,2)*
                      Math.pow(b,2) -
                  2 * Math.pow(b,4) + 4 * Math.pow(c,2) - 4 * Math.pow(a
                      ,2)* Math.pow(c,2) - 4* Math.pow(b,2) * Math.pow(c
                      ,2) -
                  2 * Math.pow(c,4))))/(2 * (-1 * Math.pow(a,2) - 2 * a *
                      b - Math.pow(b,2) - 2 * Math.pow(c,2) ));

/** /
        // 5 = { (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta]^3 - Cos[theta]
                Sin[theta]^2 +
//             \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2] Sin[theta]^2 +
//               4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 +
//               4 Sin[theta]^4 + 4 Sqrt[2] Sin[theta]^4 -
//               4 Cos[theta]^2 Sin[theta]^4 - 2 Sin[theta]^6) )) /
//             (2 (Cos[theta]^2 +
//                2 Sin[theta]^2 ) ),
        points[i][5][0] =  (cos - sqrt2 * cos - Math.pow(cos,3) - cos * Math.
              pow(sin, 2) +
          Math.sqrt( -6 * Math.pow(sin, 2) + 4 * sqrt2 * Math.pow(sin, 2) +
          4 * Math.pow(cos, 2) * Math.pow(sin , 2) - 2 * Math.pow(cos, 4) *
                Math.pow(sin, 2) +
          4 * Math.pow(sin, 4) + 4 * sqrt2 * Math.pow(sin, 4) -
          4 * Math.pow(cos, 2) * Math.pow(sin, 4) - 2 * Math.pow(sin, 6) ))/
          (2 *  (Math.pow(cos, 2) +
          2 * Math.pow(sin, 2) ));

   //     (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta]^3 - Cos[theta] Sin[
        theta]^2 +
   //        \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2] Sin[theta]^2 +
   //          4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 +
   //          4 Sin[theta]^4 + 4 Sqrt[2] Sin[theta]^4 -
   //          4 Cos[theta]^2 Sin[theta]^4 - 2 Sin[theta]^6) )) /
   //       (2 (Cos[theta]^2 +
   //          2 Sin[theta]^2 ) ),
        points[i][5][1] =  (cos - sqrt2 * cos - Math.pow(cos,3) - cos * Math.
              pow(sin, 2) +
          Math.sqrt( -6 * Math.pow(sin, 2) + 4 * sqrt2 * Math.pow(sin, 2) +
          4 * Math.pow(cos, 2) * Math.pow(sin , 2) - 2 * Math.pow(cos, 4) *
                Math.pow(sin, 2) +
          4 * Math.pow(sin, 4) + 4 * sqrt2 * Math.pow(sin, 4) -
          4 * Math.pow(cos, 2) * Math.pow(sin, 4) - 2 * Math.pow(sin, 6) ))/
          (2 *  (Math.pow(cos, 2) +
          2 * Math.pow(sin, 2) ));


//           \[Sqrt](Sqrt[2] - (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta
        ]^3 - Cos[theta] Sin[theta]^2 +
//             \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2] Sin[theta]^2 +
//             4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 + 4
        Sin[theta]^4 +
//             4 Sqrt[2] Sin[theta]^4 - 4 Cos[theta]^2 Sin[theta]^4 -
//             2 Sin[theta]^6))^2/(2 (Cos[theta]^2 + 2 Sin[theta]^2)^2))
        points[i][5][2] =
          Math.sqrt(sqrt2 - Math.pow(cos - sqrt2 * cos - Math.pow(cos, 3) -
                cos * Math.pow(sin, 2) +
          Math.sqrt(-6 * Math.pow(sin, 2) + 4 * sqrt2 * Math.pow(sin, 2) +
          4 * Math.pow(cos,2) * Math.pow(sin,2) - 2 * Math.pow(cos, 4) *
                Math.pow(sin, 2) + 4 * Math.pow(sin, 4) +
          4 * sqrt2 * Math.pow(sin, 4) - 4 * Math.pow(cos, 2) * Math.pow(sin,
                4) -
          2 * Math.pow(sin, 6)), 2) / (2 * Math.pow((Math.pow(cos, 2) + 2 *
                Math.pow(sin, 2)),2) ));
/**/
```

```

        // 5 = {               , (-1 + a^2 + b^2 + c^2 - c* z) / (a + b)
//                               , (-1 + a^2 + b^2 + c^2 - c* z) / (a + b)
//                               , (4 c - 4 a^2 c - 4 b^2 c -
//                              4 c^3 + \[Sqrt]((-4 c + 4 a^2 c + 4 b^2 c + 4 c^3)
    ^2 -
//                                       4 (-a^2 - 2 a b - b^2 - 2 c^2) (-2 + (4 +
    Sqrt[2]) a^2 -
//                                           2 a^4 + 2 Sqrt[2] a b + (4 + Sqrt[2]) b
    ^2 - 4 a^2 b^2 -
//                                           2 b^4 + 4 c^2 - 4 a^2 c^2 - 4 b^2 c^2 -
    2 c^4)))/(2 (-a^2 -
//                                     2 a b - b^2 - 2 c^2))
//                                   )                  ,
            a = points[i][4][0];
            b = points[i][4][1];
            c = points[i][4][2];

//              x=(a + a^3 + b + a^2 b + a b^2 + b^3 + a c^2 + b c^2 -
//                      Sqrt[2] Sqrt[-c^2 (1 + a^4 - 8 a b + b^4 - 6 c^2 + c^4 +
//                              2 b^2 (-1 + c^2) + 2 a^2 (-1 + b^2 + c^2))])/(2 (a
    ^2 + 2 a b +
//                         b^2 + 2 c^2))
            points[i][5][0]=(a + Math.pow(a,3) + b + Math.pow(a,2) * b + a *
                Math.pow(b,2) + Math.pow(b,3) + a * Math.pow(c,2) + b * Math
                .pow(c,2) -
               Math.sqrt(2) * Math.sqrt(-1 * Math.pow(c,2) * (1 + Math.pow(a
                   ,4) - 8 * a * b + Math.pow(b,4) - 6 * Math.pow(c,2) +
                   Math.pow(c,4) +
               2 * Math.pow(b,2) * (-1 + Math.pow(c,2)) + 2 * Math.pow(a,2)
                   * (-1 + Math.pow(b,2) + Math.pow(c,2)))))/
               (2 * (Math.pow(a,2) + 2 * a * b + Math.pow(b,2) + 2 * Math.
                   pow(c,2)));
            points[i][5][1] = points[i][5][0];
            points[i][5][2] =  Math.sqrt(2) * Math.sqrt(1 - Math.pow(points[i
                ][5][0],2));

/** /
            points[i][5][0] = (a + Math.pow(a,3) + b + Math.pow(a,2) * b + a
                * Math.pow(a,4) + Math.pow(b,3) + a * Math.pow(c,2) + b *
                Math.pow(c,2) +
                    Math.sqrt(2) * Math.sqrt(-1 *Math.pow(c,2) + 2 * Math.
                        pow(a,2) * Math.pow(c,2) - Math.pow(a,4) * Math.
                        pow(c,2) + 8 * a * b * Math.pow(c,2) +
                       2 * Math.pow(a,4) * Math.pow(c,2) - 2 * Math.pow(a
                           ,2) * Math.pow(a,4) * Math.pow(c,2) - Math.pow
                           (b,4) * Math.pow(c,2) + 6 * Math.pow(c,4) - 2
                           * Math.pow(a,2) * Math.pow(c,4) -
                       2 * Math.pow(a,4) * Math.pow(c,4) - Math.pow(c,4) )
                           )/(2 * (Math.pow(a,2) + 2 * a * b + Math.pow(a
                           ,4) + 2 * Math.pow(c,2)));
            points[i][5][1] = points[i][5][0];
            points[1][5][2] = (1 + Math.pow(a,2) + Math.pow(b,2) + Math.pow(c
                ,2) - 2 * a * points[i][5][0] - 2 * b * points[i][5][0]) /
                (2 * c);
/** /
            points[i][5][0] = (a + Math.pow(a,3) + b + Math.pow(a,2) * b + a
                * Math.pow(b,2) + Math.pow(b,3) + a * Math.pow(c,2) + b *
                Math.pow(c,2) +
                  Math.sqrt(2) * Math.sqrt(-Math.pow(c,2) + 4 * Math.pow(a
                      ,2) * Math.pow(c,2) - Math.pow(a,4) * Math.pow(c,2) +
                      12 * a * b * Math.pow(c,2) +
                      4 * Math.pow(b,2) * Math.pow(c,2) - 2 * Math.pow(a
                          ,2) * Math.pow(b,2) * Math.pow(c,2) - Math.pow(
                          b,4) * Math.pow(c,2) + 10 * Math.pow(c,4) - 2 *
                          Math.pow(a,2) * Math.pow(c,4) -
                      2 * Math.pow(b,2) * Math.pow(c,4) - Math.pow(c,6)))
                          /(2 * (Math.pow(a,2) + 2 * a * b + Math.pow(b
                          ,2) + 2 * Math.pow(c,2)));
            points[i][5][1] =  points[i][5][0];
            points[i][5][2] = Math.sqrt(2-2*Math.sqrt(points[i][5][0]));
/** /
        points[i][5][2] =  (4 * c - 4 * Math.pow(a,2) * c - 4 * Math.pow(b,2)
              * c - 4 * Math.pow(c,3) -
                    Math.sqrt(Math.pow((-4 * c + 4 * Math.pow(a,2) * c + 4 *
                        Math.pow( b,2) *c + 4 * Math.pow(c,3)),2) -
                      4 * (-1 * Math.pow(a,2) - 2 * a * b - Math.pow(b,2) -
                          2 * Math.pow(c,2)) * (-2 + (4 + Math.sqrt(2)) *
                          Math.pow(a,2) -
                        2 * Math.pow(a,4) + 2 * Math.sqrt(2) * a * b + (4
                            + Math.sqrt(2)) * Math.pow(b,2) - 4 * Math.
                            pow(a,2) * Math.pow(b,2) -
                        2 * Math.pow(b,4) + 4 * Math.pow(c,2) - 4 * Math.
                            pow(a,2) * Math.pow(c,2) - 4 * Math.pow(b,2)
                            * Math.pow(c,2) - 2 * Math.pow(c,4)))))/
                        (2 * (-1 * Math.pow(a,2) - 2 * a * b - Math.pow(b
                            ,2) - 2  * Math.pow(c,2)));

        points[i][5][0] = (-1 + Math.pow(a,2) + Math.pow(b,2) + Math.pow(c,2)
              - c* points[i][5][2]) / (a + b);
        points[i][5][1] = points[i][5][0];
/**/
        // 6 = {         0.0, COS(PI-theta), SIN(PI-theta)}
        //                == -1 * COS(theta) SIN(Theta)
        points[i][6][0] =  0.0;
        points[i][6][1] =  cos;
        points[i][6][2] =  sin;

        // 7 = {        -1.0, COS(PI-theta), SIN(PI-theta)}
        //                == -1 * COS(theta) SIN(Theta)
        points[i][7][0] = -1.0;
        points[i][7][1] =  cos;
        points[i][7][2] =  sin;

        // 8 = {        -1.0,            0.0,             0.0}
        points[i][8][0] = -1.0;
        points[i][8][1] =  0.0;
        points[i][8][2] =  0.0;

    }

    // get angles
    for (i = 0; i < numberOfLevels; i++){
        angles[i][0] = Math.PI;

        origin[0] = points[i][1][0] / 2;
        origin[1] = points[i][1][1] / 2;
        origin[2] = points[i][1][2] / 2;
        angles[i][1] = getAngle(origin, points[i][8], points[i][2]);

        angles[i][2] = getAngle(points[i][2],points[i][1], points[i][3]);

        origin[0] = points[i][3][0] / 2;
        origin[1] = points[i][3][1] / 2;
        origin[2] = points[i][3][2] / 2;
        angles[i][3] = getAngle(origin,points[i][2], points[i][4]);

        angles[i][4] = getAngle(points[i][4],points[i][3], points[i][5]);

        origin[0] = points[i][5][0] / 2;
        origin[1] = points[i][5][1] / 2;
        origin[2] = points[i][5][2] / 2;
        angles[i][5] = getAngle(origin, points[i][4], points[i][6]);

        angles[i][6] = getAngle(points[i][6],points[i][5], points[i][7]);

        origin[0] = points[i][7][0] / 2;
        origin[1] = points[i][7][1] / 2;
```

```
      origin[2] = points[i][7][2] / 2;
      angles[i][7] = getAngle(origin, points[i][6], points[i][8]);

      angles[i][8] = getAngle(points[i][8],points[i][7], points[i][1]);


    }


    // get point of the magnet
    // get unitVector of the magnet
    // get distance from magnets

    for (i = 0; i < numberOfLevels; i++){
      pointOfMagnets[i][0][0] = points[i][4][0] / 2;
      pointOfMagnets[i][0][1] = points[i][4][1] / 2;
      pointOfMagnets[i][0][2] = points[i][4][2] / 2;

      pointOfMagnets[i][1][0] = points[i][6][0] / 2;
      pointOfMagnets[i][1][1] = points[i][6][1] / 2;
      pointOfMagnets[i][1][2] = points[i][6][2] / 2;



      unitVectorOfMagnets[i][0]= new Vector();
      unitVectorOfMagnets[i][1]= new Vector();

      v1 = new Vector();
      v2 = new Vector();


      v1.setXYZ((float)points[i][3][0],
          (float)points[i][3][1],
          (float)points[i][3][2] );
      v1.invert();

      unitVectorOfMagnets[i][0].setXYZ((float)points[i][4][0],
          (float)points[i][4][1],
          (float)points[i][4][2] );

      unitVectorOfMagnets[i][0].addVector(v1);
      unitVectorOfMagnets[i][0] = unitVectorOfMagnets[i][0].getUnitVector()
          ;

      v2.setXYZ((float)points[i][7][0],
          (float)points[i][7][1],
          (float)points[i][7][2] );
      v2.invert();

      unitVectorOfMagnets[i][1].setXYZ((float)points[i][6][0],
          (float)points[i][6][1],
          (float)points[i][6][2] );
      unitVectorOfMagnets[i][1].addVector(v2);
      unitVectorOfMagnets[i][1] = unitVectorOfMagnets[i][1].getUnitVector()
          ;

      distancesBetweenMagnets[i] = getDistance(points[i][4], points[i][6])
          / 2;

    }


    FileBoxCornerAngleData fileBoxCornerAngleData = new
         FileBoxCornerAngleData();
    if (fileBoxCornerAngleData.build(FILENAME, points, angles,
         pointOfMagnets, unitVectorOfMagnets, distancesBetweenMagnets)){

      System.out.format("%s is created\n", FILENAME);

    } else {
      System.out.format("error during %s is being creating\n", FILENAME);

    }

    MonitorBoxCornerSimulator monitor = new MonitorBoxCornerSimulator ();
    monitor.run(points);

  }

  private double getAngle(double [] origin, double [] v1, double[] v2) {
    int i;
    double angle = 0;
    double dot = 0;

    double [] uv1 = new double [3];
    double [] uv2 = new double [3];

    double uv1Length = 0;
    double uv2Length = 0;


    for(i=0; i<3 ; i++) {
      uv1[i] = v1[i] - origin[i] ;
      uv2[i] = v2[i] - origin[i] ;

      uv1Length += Math.pow(uv1[i], 2);
      uv2Length += Math.pow(uv2[i], 2);
    }


    uv1Length = Math.sqrt(uv1Length);
    uv2Length = Math.sqrt(uv2Length);

    dot = 0;
    for(i=0; i<3 ; i++) {
      uv1[i] = uv1[i] / uv1Length;
      uv2[i] = uv2[i] / uv2Length;

      dot += uv1[i] * uv2[i];
    }

    if (dot>1) {
      dot = 1;
    }
    if (dot<-1) {
      dot = -1;
    }

    angle = Math.acos(dot);
    return angle;
  }

  public double getDistance(double v1[], double v2[]) {
      return Math.sqrt(Math.pow(v2[0]- v1[0], 2) + Math.pow(v2[1]- v1[1],
           2) + Math.pow(v2[2]- v1[2] ,2));
  }


  public static void main(String[] args) {



    BoxCornerSimulator boxCornerSimulator = new BoxCornerSimulator();
    boxCornerSimulator.run();

  }
}
```

```java
package com.drancom.programmableMatter.folding.simulator.boxcorner;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;

import com.drancom.programmableMatter.folding.controller.paper.util.Vector;

public class FileBoxCornerAngleData {
  public boolean build(String fileName, double [][][] points, double [][]
        angles, double[][][] pointOfMagnets, Vector [][] unitVectorOfMagnets
         ,double [] distantBetweenMagnets ) {
    int i;
    int j;
    int k;


    int numberOfLevels;
    int numberOfEdges = 8;


    File file = new File(fileName);

    numberOfLevels = points.length;

    try {
      boolean success = file.createNewFile();
      if (success) {
        // File did not exist and was created
      } else {

      }
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    String bufferLine  = new String();

    try {
      BufferedWriter bufferedWriter = new BufferedWriter (new FileWriter (
           file));

      // print points
      bufferLine = String.format("# %d levels", numberOfLevels);
      // print to file
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      bufferLine = String.format("# %d edges", 8);
      // print to file
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      for (i = 0; i < numberOfLevels; i++) {

        for (j=0 ; j < 9 ; j++) {
          bufferLine = String.format("p");
          bufferLine += String.format(", %d, %f, %f, %f", j, points[i][j
              ][0]
                                                          , points[i][j
                                                            ][1]
                                                          , points[i][j
                                                            ][2]);

          // print to file
          bufferedWriter.write(bufferLine);
          bufferedWriter.newLine();
        }
      }

      // print angles
      bufferLine = String.format("# %d levels", numberOfLevels);

      // print to file
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      bufferLine = String.format("# %d edges", 8);
      // print to file
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      // angle[level][edgeNumber]
      for (i = 0; i < numberOfLevels; i++) {
        for (j = 1; j < numberOfEdges + 1; j++) {
          bufferLine = String.format("a");
          bufferLine += String.format(", %d, %f", j, angles[i][j]);

          // print to file
          bufferedWriter.write(bufferLine);
          bufferedWriter.newLine();
        }
      }

      bufferLine = String.format("# %d edges", 8);
      // print to file
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      // angle[level][edgeNumber]
      for (i = 0; i < numberOfLevels; i++) {
        bufferLine = String.format("#g");
        bufferLine += String.format(", %d", i);
        for (j = 1; j < numberOfEdges + 1; j++) {

          bufferLine += String.format(", %f", angles[i][j]);


        }
        // print to file
        bufferedWriter.write(bufferLine);
        bufferedWriter.newLine();      }


      bufferLine = String.format("# pointOfMagnet" );
      // print to file
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      bufferLine = String.format("# Magnet1'sX,Y,Z,Magnet2'sX,Y,z" );
      // print to file
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      // point of magnet
      for (i = 0; i < numberOfLevels; i++) {
        bufferLine = String.format("#pm");
        bufferLine += String.format(", %d", i);
        for (j=0; j<2; j++) {
          for (k=0;k < 3;k++){
            bufferLine += String.format(", %f", pointOfMagnets[i][j][k]);
          }
        }
        // print to file
        bufferedWriter.write(bufferLine);
        bufferedWriter.newLine();
      }
      bufferLine = String.format("# UnitVectorOfMagnet1'sX,Y,Z,
           UnitVectorOfMagnet2'sX,Y,z" );
      // print to file
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      // unitVector of magnet
      for (i = 0; i < numberOfLevels; i++) {
        bufferLine = String.format("#uv");
        bufferLine += String.format(", %d", i);
        for (j=0; j<2; j++) {
          bufferLine += String.format(", %f", unitVectorOfMagnets[i][j].
               getX());
          bufferLine += String.format(", %f", unitVectorOfMagnets[i][j].
               getY());
          bufferLine += String.format(", %f", unitVectorOfMagnets[i][j].
               getZ());
        }
        // print to file
        bufferedWriter.write(bufferLine);
        bufferedWriter.newLine();
      }


      bufferLine = String.format("# %d distant", 1);
      // print to file
      bufferedWriter.write(bufferLine);
      bufferedWriter.newLine();

      // distants level]
      for (i = 0; i < numberOfLevels; i++) {
        bufferLine = String.format("#d");
        bufferLine += String.format(", %d", i);
        bufferLine += String.format(", %f", distantBetweenMagnets[i]);


        // print to file
        bufferedWriter.write(bufferLine);
        bufferedWriter.newLine();
      }

      // file close
      bufferedWriter.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
      return false;
    } catch (IOException e) {
      e.printStackTrace();
      return false;
    }
    return true;
  }
}
```

```java
package com.drancom.programmableMatter.folding.simulator.boxcorner;


import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.*;

import javax.media.opengl.*;

import sun.text.normalizer.UProperty;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.Point;
import com.drancom.programmableMatter.folding.controller.paper.Polygon;
import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.sun.opengl.util.*;

public class MonitorBoxCornerSimulator implements GLEventListener,
     MouseListener, MouseMotionListener {
  public final static double ZOOM_MAGNIFICATION = 4;
  public final static float LINEWIDTH = 2;

  double [][][] points;

  // light
  // float pos0[] = {  -100.0f,  130.0f, 150.0f,  1.0f };
  public final static float pos0[] = {  -100.0f,  130.0f, 150.0f,  1.0f };
  public final static float ambientLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
  public final static float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
  public final static float specular[] = { 0.25f, 0.25f, 0.25f, 1.0f};

  public final static float WHITE[]  = {  1.0f,  1.0f,  1.0f,  0.2f };
  public final static float RED[]    = {  1.0f,  0.0f,  0.0f,  1.0f };
  public final static float GREEN[]  = {  0.0f,  1.0f,  0.0f,  1.0f };
  public final static float YELLOW[] = {  1.0f,  1.0f,  0.0f,  1.0f };
  public final static float BLUE[]   = { 0.0f, 0.0f, 1.0f, 0.0f };
  public final static float BLACK[]  = { 0.0f, 0.0f, 0.0f, 1.0f };

  final static int SPEED_OF_ANIMATION = 10; //10 is default
  int counterForSpeedOfAnimation = 0;

  boolean isAnimating=false;

  public void run(double[][][] points) {
    this.points = points;
    paperGlId = new int[points.length];


    Frame frame = new Frame("Play Window - Programmable Matter by Folding")
         ;
      GLCanvas canvas = new GLCanvas();

    canvas.addGLEventListener(this);
    frame.add(canvas);
    frame.setSize(800, 800);
    final Animator animator = new Animator(canvas);
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
            // Run this on another thread than the AWT event queue to
        // make sure the call to Animator.stop() completes before
        // exiting
        new Thread(new Runnable() {
          public void run() {
            animator.stop();
            System.exit(0);
          }
        }).start();
          }
    });
      frame.show();
      animator.start();
  }

  private float view_rotx =  -20.0f, view_roty =  -10.0f, view_rotz = 10.0f
       ;

  private int paperGlId[];
  private int currentIndex_PaperGlId;

  private boolean isFolding = true;

  public boolean isFolding() {
    return isFolding;
  }

  private int prevMouseX, prevMouseY;
  private boolean mouseRButtonDown = false;

  public void init(GLAutoDrawable drawable) {
    int i;

    // Use debug pipeline
    // drawable.setGL(new DebugGL(drawable.getGL()));

    GL gl = drawable.getGL();

    System.err.println("INIT GL IS: " + gl.getClass().getName());
    System.err.println("Chosen GLCapabilities: " + drawable.
         getChosenGLCapabilities());

    gl.setSwapInterval(1);

    // Blend
    gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE);
    gl.glEnable(GL.GL_BLEND);


//      gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
//        gl.glClearDepth(1.0f);

    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos0, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambientLight, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuseLight, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular, 0);

    gl.glEnable(GL.GL_CULL_FACE);

    gl.glEnable(GL.GL_LIGHTING);
      gl.glEnable(GL.GL_LIGHT0);

//      gl.glEnable(GL.GL_DEPTH_TEST);

      /* make the papers */
      for (i=0; i<points.length; i++){
        paperGlId[i] = gl.glGenLists(1);
        gl.glNewList(paperGlId[i], GL.GL_COMPILE);
        buildGlPaper(gl, points[i]);
        gl.glEndList();
      }

    currentIndex_PaperGlId = 0; // start with folding
      // currentIndex_PaperGlId = points.length -1; //start with unfolding

    gl.glEnable(GL.GL_NORMALIZE);

      drawable.addMouseListener(this);
      drawable.addMouseMotionListener(this);
  }

  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
        height) {
    GL gl = drawable.getGL();

    float h = (float)height / (float)width;

    gl.glMatrixMode(GL.GL_PROJECTION);

    System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
    System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
    System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
    gl.glLoadIdentity();
    gl.glFrustum(-1.0f, 1.0f, -h, h, 2.0f, 30.0f);
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0.0f, 0.0f, -20.0f);
  }

  public void display(GLAutoDrawable drawable) {
    // Turn the gears' teeth

    // Get the GL corresponding to the drawable we are animating
    GL gl = drawable.getGL();

    // Special handling for the case where the GLJPanel is translucent
    // and wants to be composited with other Java 2D content
    if ((drawable instanceof GLJPanel) &&
        !((GLJPanel) drawable).isOpaque() &&
        ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
      gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
    } else {
      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    }


    // Rotate the entire assembly of gears based on how the user
    // dragged the mouse around
    gl.glPushMatrix();
    gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
    gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
    gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);

    // Place the first gear and call its display list
    gl.glPushMatrix();
      gl.glTranslatef(0.0f, 0.0f, 0.0f);
      gl.glCallList(paperGlId[((int) currentIndex_PaperGlId)]);
    gl.glPopMatrix();


    // Remember that every push needs a pop; this one is paired with
    // rotating the entire gear assembly
    gl.glPopMatrix();
    if (isAnimating == true){
      if (isFolding() == true ){
        counterForSpeedOfAnimation--;

        if (counterForSpeedOfAnimation <= 0) {
          counterForSpeedOfAnimation = SPEED_OF_ANIMATION;

          currentIndex_PaperGlId--;

          if (currentIndex_PaperGlId < 0) {
            currentIndex_PaperGlId = 0;
            isFolding = false;
            isAnimating = false;
          }
        }
      } else {
        counterForSpeedOfAnimation++;
        if (counterForSpeedOfAnimation >= SPEED_OF_ANIMATION) {
          counterForSpeedOfAnimation = 0;

          currentIndex_PaperGlId++;

          if (currentIndex_PaperGlId >= points.length - 1 ) {
            currentIndex_PaperGlId = points.length - 1;
            isFolding = true;
            isAnimating = false;
          }
        }
      }
    }
  }

  public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
       boolean deviceChanged) {}

  public void buildGlPaper(GL gl, double [][] points){
    int i;
    int j;
    int numberOfPoint = points.length;


    gl.glShadeModel(GL.GL_FLAT);

    gl.glNormal3f(0.0f, 0.0f, 1.0f);

    /* draw polygon */
    for (i=0; i<points.length - 1; i++){

      gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WHITE, 0);

      gl.glBegin(GL.GL_TRIANGLES);
      gl.glVertex3d(points[0][0] * ZOOM_MAGNIFICATION
             , points[0][1] * ZOOM_MAGNIFICATION
             , points[0][2] * ZOOM_MAGNIFICATION);

      gl.glVertex3d(points[1 + i%8][0] * ZOOM_MAGNIFICATION
             , points[1 + i%8][1] * ZOOM_MAGNIFICATION
             , points[1 + i%8][2] * ZOOM_MAGNIFICATION);

      gl.glVertex3d(points[1 + (i + 1)% 8][0] * ZOOM_MAGNIFICATION
             , points[1 + (i + 1)% 8][1] * ZOOM_MAGNIFICATION
             , points[1 + (i + 1)% 8][2] * ZOOM_MAGNIFICATION);
/**/
      gl.glVertex3d(points[0][0] * ZOOM_MAGNIFICATION
          , points[0][1] * ZOOM_MAGNIFICATION
          , points[0][2] * ZOOM_MAGNIFICATION);

      gl.glVertex3d(points[1 + (i + 1)% 8][0] * ZOOM_MAGNIFICATION
             , points[1 + (i + 1)% 8][1] * ZOOM_MAGNIFICATION
             , points[1 + (i + 1)% 8][2] * ZOOM_MAGNIFICATION);

      gl.glVertex3d(points[1 + i%8][0] * ZOOM_MAGNIFICATION
          , points[1 + i%8][1] * ZOOM_MAGNIFICATION
          , points[1 + i%8][2] * ZOOM_MAGNIFICATION);
/**/
      gl.glEnd();

    }

    /* draw lines */

    for (i=0; i<points.length-1; i++) {

        gl.glLineWidth(LINEWIDTH);



      gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WHITE, 0);

      gl.glBegin(GL.GL_LINES);

      gl.glVertex3d((float)points[0][0]*ZOOM_MAGNIFICATION,
          (float)points[0][1]*ZOOM_MAGNIFICATION,
          (float)points[0][2]*ZOOM_MAGNIFICATION);

      gl.glVertex3d(points[1 + i%8][0] * ZOOM_MAGNIFICATION
          , points[1 + i%8][1] * ZOOM_MAGNIFICATION
          , points[1 + i%8][2] * ZOOM_MAGNIFICATION);

      gl.glVertex3d(points[1 + i%8][0] * ZOOM_MAGNIFICATION
          , points[1 + i%8][1] * ZOOM_MAGNIFICATION
          , points[1 + i%8][2] * ZOOM_MAGNIFICATION);

      gl.glVertex3d(points[1 + (i + 1)% 8][0] * ZOOM_MAGNIFICATION
          , points[1 + (i + 1)% 8][1] * ZOOM_MAGNIFICATION
          , points[1 + (i + 1)% 8][2] * ZOOM_MAGNIFICATION);

      gl.glEnd();


    }
  }

  // Methods required for the implementation of MouseListener
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {

    prevMouseX = e.getX();
    prevMouseY = e.getY();
    if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
      mouseRButtonDown = true;
    }

    if (mouseRButtonDown == true) {
      isAnimating = true;
    }
  }

  public void mouseReleased(MouseEvent e) {
    if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
      mouseRButtonDown = false;
      }
  }

  public void mouseClicked(MouseEvent e) {}

  // Methods required for the implementation of MouseMotionListener
  public void mouseDragged(MouseEvent e) {
    if (mouseRButtonDown == false) {
      int x = e.getX();
      int y = e.getY();
      Dimension size = e.getComponent().getSize();

      float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
      float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);

      prevMouseX = x;
      prevMouseY = y;

      view_rotx += thetaX;
      view_roty += thetaY;
    }else {

    }
  }

  public void mouseMoved(MouseEvent e) {

  }
}
```

```
package com.drancom.programmableMatter.folding.simulator.
     simulatorForOrigamiWithStepFunction;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FilePlan;
import com.drancom.programmableMatter.folding.origami.planner.Plan;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
        .FilePlanForOrigami;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
        .PlanForOrigami;
import com.drancom.programmableMatter.stepFunction.StepFunction;

public class PlanerForOrigamiWithStepFunction implements Planner {
//  Default: percentage of level0 = 33, percentage of level1 = 66,
      percentage of level2 = 100
//  NoiseNumber = 0.01
  final static float EPS = 0.5f;

  final static float PERCENTAGE_OF_LEVEL0 = 0.33f;
  final static float PERCENTAGE_OF_LEVEL1 = 0.66f;
  final static float PERCENTAGE_OF_LEVEL2 = 1.0f;

  final static float NOISE_NUMBER= 0.01f;

  PlanForOrigami planForOrigami;

  @Override
  public void build(Paper[] papers) {

//      Input:  Angledata[numberOfEdge][time]
    float angleData[][] = new float[papers[0].getNumberOfEdges()][papers.
          length];

//      Output: Plan[phase][n], numberOfPhase
    this.planForOrigami = new PlanForOrigami();
    int plan[][] = new int[papers.length][papers[0].getNumberOfEdges()];
    int numberOfPhases;

    float angleLevel[][] = new float [papers[0].getNumberOfEdges()][papers.
          length];
    int angleDifference[][] = new int [papers[0].getNumberOfEdges()][papers
          .length];

    int phase;
    boolean isBuildingPlan;
    boolean isAllZero;

    int i;
    int j;
    int k=0;

    for (i=0; i < papers[0].getNumberOfEdges(); i++){
      for (j=0; j < papers.length; j++) {
        angleData[i][j] = papers[j].getLine(i).getAngle();
        angleLevel[i][j] = 0;
        if (angleData[i][j] <0) {
          k++;
        }
      }
    }


//      1-6 StepFunction
//      1. for i = 0 to NumberOfEdge
//      2.      angleLevel[i] <- stepFunction(angledata[i], EPS) // EPS = 0.5
    for(i=0; i < papers[0].getNumberOfEdges(); i++) {
      angleLevel[i] =  StepFunction.stepFunction(angleData[i], EPS);
    }

//      3. for i = 0 to numberOfEdge
//      4.      for j=0 to papers.length -1
//      5.         angleDifference [i][j] =0;
//      6.         if (angleLevel[i][j] > 0 && angleLevel[i][j] != angleLevel[i
      ][0] && angleLevel[i][j] != angleLevel[i][angleLevel[i].length - 1 ])
//      7.            angleDifference[i][j] = 1;
//      8.         if (angleLevel[i][j] > 0 && angleLevel[i][j] - angleLevel[i][
      j+1] != 0 ) {
//      9.            angleDifference[i][j] = 1;
//      10.        if (angleLevel[i][j] < 0 && angleLevel[i][j] != angleLevel[i
      ][0] && angleLevel[i][j] != angleLevel[i][angleLevel[i].length - 1 ])
//      11.           angleDifference[i][j] = -1;
//      12.        if (angleLevel[i][j] < 0 && angleLevel[i][j] - angleLevel[i][
      j+1] != 0 ) {
//      13.           angleDifference[i][j] = -1;

    for (i=0; i < papers[0].getNumberOfEdges(); i++){
      for (j=0; j < papers.length-1; j++) {

        angleDifference[i][j] = 0;
        if (angleLevel[i][j] > 0 && angleLevel[i][j] != angleLevel[i][0] &&
              angleLevel[i][j] != angleLevel[i][angleLevel[i].length - 1 ])
            {
          angleDifference[i][j] = 1;
        }
        if (angleLevel[i][j] > 0 && angleLevel[i][j] - angleLevel[i][j+1]
            != 0 ) {
          angleDifference[i][j] = 1;
        }
        if (angleLevel[i][j] < 0 && angleLevel[i][j] != angleLevel[i][0] &&
              angleLevel[i][j] != angleLevel[i][angleLevel[i].length - 1 ])
            {
          angleDifference[i][j] = -1;
        }
        if (angleLevel[i][j] < 0 && angleLevel[i][j] - angleLevel[i][j+1]
            != 0 ) {
          angleDifference[i][j] = -1;
        }
      }
    }

//      14. Phase <- 0
//      15. numberOfPhase
//      16. isBuildingPlan <- false
//      17. IsAllZero <- true
    phase = 0;
    numberOfPhases = 1;
    isBuildingPlan = false;
    isAllZero = true;

//      18. Plan[1..][1..numberOfEdge] <- 0
    for (i=0; i < papers.length; i++) {
      for (j=0; j < papers[0].getNumberOfEdges(); j++){
        plan[i][j] = 0;
      }
    }

//      19. For i = lastTime to 1
    for (i=papers.length -1 ; i>=0; i--) {


//      20.         IsAllZero = true
      isAllZero = true;
//      21.         For j = 1 to numberOfEdge
      for (j=0; j< papers[0].getNumberOfEdges(); j++) {
//      22.                 If AngleDifference[j][i] = -1,
        if (angleDifference[j][i] == -1){
//      23.                          Do Plan[Phase][j] = -1
//      24.                          isBuildingPlan = true
//      25.                          IsAllZero = false
          plan[phase][j] = -1;
          isBuildingPlan = true;
          isAllZero = false;
        }
//      26.                 If AngleDifference[j][i] = 1,
          if (angleDifference[j][i] == 1){
//      27.                      Do Plan[Phase][j] = 1
//      28.                      isBuildingPlan = true
//      29.                      IsAllZero = false
            plan[phase][j] =  1;
            isBuildingPlan = true;
            isAllZero = false;
          }
        }
//      30.         If isBuildingPlan = true and isAllzero = true
        if (isBuildingPlan == true && isAllZero == true) {
//      31.               Do lastPhase = Phase
//      32.               Phase++
//      33.               isBuildingPlan = false
          phase++;
          numberOfPhases  = phase;
          isBuildingPlan = false;
        }
      }

      int temp_plan[][] = new int[numberOfPhases][papers[0].getNumberOfEdges
          ()];
      for (i = 0; i < numberOfPhases; i++) {
        for(j=0; j < papers[0].getNumberOfEdges() ; j++){
          temp_plan[i][j] = plan[i][j];
        }
      }

      // build edge table
      float edgeTable[][] = new float[papers[0].getNumberOfEdges()][4];
      for (i=0 ; i < papers[0].getNumberOfEdges(); i++) {
        edgeTable[i][0] = (float) papers[0].getLine(i).getStartPoint().
            getXOnPaper();
        edgeTable[i][1] = (float) papers[0].getLine(i).getStartPoint().
            getYOnPaper();
        edgeTable[i][2] = (float) papers[0].getLine(i).getEndPoint().
            getXOnPaper();
        edgeTable[i][3] = (float) papers[0].getLine(i).getEndPoint().
            getYOnPaper();
      }

      this.planForOrigami.setEdgeTable(edgeTable);
      this.planForOrigami.setPlanTable(temp_plan);
      this.planForOrigami.setNumberOfEdges(papers[0].getNumberOfEdges());
      this.planForOrigami.setNumberOfPhases(numberOfPhases);
    }

    @Override
    public void exportPlan(String fileName, Paper[] papers) {

    }

    @Override
    public Plan getPlan() {
      // TODO Auto-generated method stub
      return this.planForOrigami;
    }

    @Override
    public void build(Plan[] plans) {
      // TODO Auto-generated method stub

    }

    @Override
    public void exportPlan(String fileName) {
      FilePlan filePlan = new FilePlanForOrigami();

      filePlan.build(fileName, planForOrigami);
    }
}
```

```
package com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigamiWithStepFunction;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.origami.planner.
      PlannerForWiring;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
    .MainWindowForFoldingRobotOrigami;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
    .PlanForOrigami;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
    .PlanerForOrigami;
import com.drancom.programmableMatter.stepFunction.StepFunction;

public class SimulatorForOrigamiWithStepFunction {

  /** /
    public static final String PLAN_FILENAME = "V:\\com\\dran\\vc\\pm\\
          RigidOrigami006\\RigidOrigami\\save_8x8_s-shuttle\\
          plan_for_origami_s-shuttle.csv";
    public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
          RigidOrigami006\\RigidOrigami\\save_8x8_s-shuttle\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 50;
  /**/
    public static final String PLAN_FILENAME = "V:\\com\\dran\\vc\\pm\\
          RigidOrigami006\\RigidOrigami\\save_8x8_hat\\
          plan_for_origami_8x8_hat.csv";
    public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
          RigidOrigami006\\RigidOrigami\\save_8x8_hat\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 28;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_8x8airplain\\plan_for_origami_8x8airplain.csv";
    public static final String FILENAME = "c:\\foldingdata\\
          save_8x8airplain\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 50;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_8x8box\\plan_for_origami_8x8box.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_8x8box\\m
          %05d.obj";
    public static final int NUMBER_OF_FILES= 70;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_8x8sailboat\\plan_for_origami_8x8sailboat.csv";
    public static final String FILENAME = "c:\\foldingdata\\
          save_8x8sailboat\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 50;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_8x8bench\\plan_for_origami_save_8x8bench.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_8x8bench\\
          m%05d.obj";
    public static final int NUMBER_OF_FILES= 70;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_8x8table\\plan_for_origami_save_8x8table.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_8x8table\\
          m%05d.obj";
```

```java
    public static final int NUMBER_OF_FILES= 30;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_8x8table_b\\plan_for_origami_save_8x8table_b.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_8x8table_b
          \\m%05d.obj";
    public static final int NUMBER_OF_FILES= 25;

  /**/

//      public static final String FILENAME = "c:\\foldingdata\\save_box\\
      m00070.obj";
//      public static final int NUMBER_OF_FILES= 1;

    Paper[] papers;
    FileObj[] fileObjs;
    MainWindowForFoldingRobotOrigami mainWindow;
    public SimulatorForOrigamiWithStepFunction() {

    }

    void run() {
      int i;
      String fileName;

      // Initiation
      papers = new Paper[NUMBER_OF_FILES];
      fileObjs = new FileObj[NUMBER_OF_FILES];
      mainWindow = new MainWindowForFoldingRobotOrigami();


      // load
      for (i=0; i < NUMBER_OF_FILES; i++) {
        fileName = String.format(FILENAME, i+1);
        papers[i] = new Paper();
        fileObjs[i] = new FileObj();
        fileObjs[i].load(fileName, papers[i]);
        papers[i].sortLine();
      }

      for (i=0; i < papers[0].getNumberOfEdges(); i++) {
        System.out.printf("%d, %f, %f, %f, %f\n" , i, papers[3].getLine(i).
              getStartPoint().getXOnPaper()
            , papers[0].getLine(i).getStartPoint().getYOnPaper()
            , papers[0].getLine(i).getEndPoint().getXOnPaper()
            , papers[0].getLine(i).getEndPoint().getYOnPaper());
      }
      // sort paper


      Planner planer = new PlanerForOrigamiWithStepFunction(); // new
            planer

      planer.build(papers);

      planer.exportPlan(PLAN_FILENAME);

      mainWindow.run(papers, (PlanForOrigami) planer.getPlan());

    }

    public static void main(String[] args) {
      // TODO Auto-generated method stub
      SimulatorForOrigamiWithStepFunction simulator = new
            SimulatorForOrigamiWithStepFunction ();
      simulator.run();
    }
}
```

```java
package com.drancom.programmableMatter.folding.simulator.
      simulatorForPlanOfOrigami;

import com.drancom.programmableMatter.folding.controller.paper.Line;
import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.Point;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
      .PlanForOrigami;
import com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigamiWithStepFunction.PlanerForOrigamiWithStepFunction;

public class FilterOfPlanOfAlgorithm {
  public boolean filtering(Paper paper, PlanForOrigami plan) {
    if (paper.getNumberOfEdges() != plan.getNumberOfEdges()) {
      return false;
    }

    int i, j, k;
    Line line;
    Point startPoint = new Point();
    Point endPoint   = new Point();

    float edgeTable [][] ;
    int planTable [][];

    edgeTable = plan.getEdgeTable();
    planTable = plan.getPlanTable();
    int numberOfLine = 0;
    boolean isAllZero;
    for (i=0; i< edgeTable.length; i++) {
      isAllZero = true;
      for (j = 0; j < planTable.length; j++) {
        if (planTable[j][i] != 0 ) {
          isAllZero = false;
        }
      }
      if (isAllZero) {
        startPoint = paper.getPoint(edgeTable[i][0], edgeTable[i][1], 0.0f)
              ;
        endPoint = paper.getPoint(edgeTable[i][2], edgeTable[i][3], 0.0f);

        line = paper.getLine(startPoint, endPoint);
        line.setAngle(0.0f);
      }

    }



    return true;
  }
}
```

```java
package com.drancom.programmableMatter.folding.simulator.
      simulatorForPlanOfOrigami;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.MainWindow;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.origami.planner.
      PlannerForWiring;
import com.drancom.programmableMatter.folding.simulator.Simulator;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
      .PlanForOrigami;
import com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigamiWithStepFunction.PlanerForOrigamiWithStepFunction;

```

```java
public class SimulatorOfPlanOfOrigami {

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_airplain\\plan_airplain.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_airplain\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 50;

  /**/
    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_box\\plan_box.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 70;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_sailboat2\\plan_sailboat2.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_sailboat2\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 35;

  /** /
    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_8x8bench\\plan_save_bench.csv";
    public static final String FILENAME = "c:\\foldingdata\\save_8x8bench\\m%05d.obj";
    public static final int NUMBER_OF_FILES= 70;

  /**/

//      public static final String FILENAME = "c:\\foldingdata\\save_box\\m00070.obj";
//      public static final int NUMBER_OF_FILES= 1;

    Paper[] papersWithPlan;
    Paper[] papersOriginal;
    FileObj[] fileObjs;
    MainWindow mainWindow;
    public SimulatorOfPlanOfOrigami() {
    }

    void run() {
      int i;
      String fileName;

      // init
      papersWithPlan = new Paper[NUMBER_OF_FILES];
      papersOriginal = new Paper[NUMBER_OF_FILES];
      fileObjs = new FileObj[NUMBER_OF_FILES];
      mainWindow = new MainWindow();

      // load
      for (i=0; i < NUMBER_OF_FILES; i++) {
        fileName = String.format(FILENAME, i+1);
        papersOriginal[i] = new Paper();
        papersWithPlan[i] = new Paper();
        fileObjs[i] = new FileObj();
        fileObjs[i].load(fileName, papersOriginal[i]);
        fileObjs[i].load(fileName, papersWithPlan[i]);
        }



      Planner planer = new PlanerForOrigamiWithStepFunction();

      planer.build(papersOriginal);


      FilterOfPlanOfAlgorithm filterOfPlanOfAlgorithm = new FilterOfPlanOfAlgorithm();

      for (i=0; i < NUMBER_OF_FILES; i++) {
        filterOfPlanOfAlgorithm.filtering(papersWithPlan[i], (PlanForOrigami) planer.getPlan());
      }

      System.out.print(getAveDistanceDifference(papersOriginal, papersWithPlan));




      planer.exportPlan(PLAN_FILENAME, papersOriginal);

      mainWindow.run(papersOriginal);

    }

    protected float getAveDistanceDifference(Paper[] papers0, Paper[] papers1){

      float aveDistanceDifference;

      int i, j;

      Vector v0, v1;
      aveDistanceDifference = 0.0f;
      for( i = 0; i<papers0.length; i++) {
        for( j =0 ; j< papers0[i].getNumberOfPoints(); j++) {

          v0 = papers0[i].getPoint(j).getVectorInReal();
          v1 = papers1[i].getPoint(j).getVectorInReal();

          v1.invert();
          v0.addVector(v1);

          aveDistanceDifference += (float) (Math.sqrt( v0.getX()*v0.getX()
            + v0.getY()*v0.getY()
            + v0.getZ()*v0.getZ() ) ) ;

        }
      }

      System.out.print(aveDistanceDifference );
      aveDistanceDifference /=  ((double) papers0[0].getNumberOfPoints());



      return aveDistanceDifference;
    }

    public static void main(String[] args) {
      // TODO Auto-generated method stub
      SimulatorOfPlanOfOrigami simulator = new SimulatorOfPlanOfOrigami();
      simulator.run();
    }
}
```

```java
package com.drancom.programmableMatter.stepFunction;

public class StepFunction {
  public static float[] stepFunction (float [] P, float eps){
//      StepFtn[P_(* list of y-
//              value at each time step--starting at time 1 for convenience*),
//              eps_(* error tolerance*)] :=
//          Module[{Steps = {}(*list of steps*), stepNo (*total # of steps*)
      , i,
//            j, k, EBars = {}(*list with error bar at each time step*), n},

    float [] Steps = new float [P.length];
    int NumberOfSteps;
    int stepNo;
    int i;
    int j;
    int k;
    float [][] EBars = new float [P.length][2];
    int n;

    float [] intNew = new float [2];
    float [] intOld = new float [2];
    float val;
//          stepNo = 1;
//          n = Length[P];
    NumberOfSteps = 0;
    stepNo=1;
    n = P.length;
//          (*If n<1 then return NULL WRITE CODE*)
//          (*create list of error bars from P and eps*)
//          For[k = 1, k <= n, k++,
//           AppendTo[EBars, {P[[k]] - eps, P[[k]] + eps}]
//           ];

    for (k = 0; k < n; k++) {
      EBars[k][0] = P[k] - eps;
      EBars[k][1] = P[k] + eps;
    }
//          i = 1;
//          j = i + 1;
    i = 0;
    j = i + 1;
//          (*Sweep left to right maintaining intersection of the error
      bars,
//          and once empty, start new step*)
//          intOld = EBars[[1]];
//          intNew = intOld;
    intOld[0] = EBars[0][0];
    intOld[1] = EBars[0][1];

    intNew[0] = intOld[0];
    intNew[1] = intOld[1];

//          While[j <= n,
//           (*set intNew to intersection of intOld and new point's error
      bar*)
    while(j < n) {
//
//
//           intNew = {Max[intOld[[1]], EBars[[j]][[1]]],
//             Min[intOld[[2]], EBars[[j]][[2]]]};

      intNew[0] = intOld[0]>EBars[j][0]?intOld[0]:EBars[j][0];
      intNew[1] = intOld[1]<EBars[j][1]?intOld[1]:EBars[j][1];

//           If[intNew[[1]] <= intNew[[2]] && j < n(*If intersectn non-
//            empty continue with next point& not at last point*),
//              intOld = intNew;
//                 j++;,
      if (intNew[0] <= intNew[1] && j < n - 1 ) {
        intOld[0] = intNew[0];
        intOld[1] = intNew[1];
        j++;
//            (*else if intersection is empty--start new step *)
//            (*val=step value*)
//            If[j == n,
      } else if(j == n - 1) {
//                  j++;(*annoying boundary condition*)
//                val = intOld[[1]] + (intOld[[2]] - intOld[[1]])/2 ;
//                AppendTo[Steps,
//                 Line[{{i - 1/2, val}, {j, val}}]];,(*else just do this*)
        j++;
        val = intOld[0] + (intOld[1] - intOld[0]) / 2;
        for(k=i; k<j ; k++){
          Steps[NumberOfSteps] = val;
          NumberOfSteps++;
        }

      } else {
//                val = intOld[[1]] + (intOld[[2]] - intOld[[1]])/2 ;
//                AppendTo[Steps, Line[{{i - 1/2, val}, {j - 1/2, val}}]];
//                i = j;
//                j++;
//                stepNo++;
//                (*set intNew to new points error bar in preparation for
      start \
//          of the next step*)
//                intNew = EBars[[i]];
//                intOld = intNew;
//                ];
        val = intOld[0] +(intOld[1] - intOld[1]) /2;
        for(k = i; k < j ; k++){
          Steps[NumberOfSteps] = val;
          NumberOfSteps++;
        }

        i = j;
        j++;

        stepNo++;

        intNew[0] = EBars[i][0];
        intNew[1] = EBars[i][1];

        intOld[0] = intNew[0];
        intOld[1] = intNew[1];

      }
//
//               ];
//              ];
//             Print["# of Steps = ", stepNo];
//             Print[" "];
//
//             Show[{Graphics[Steps], ListPlot[P]}, Axes -> True,
//              AxesOrigin -> {0, 0}]
//
//             ]
    }
    return Steps;
  }
}
```

```
package com.drancom.programmableMatter.tile.clipSMA;

import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
      .PlanForOrigami;

public class ProgrammableMatter {
  final static float POINTS[][][] =
      {{{0.0f, 0.0f}, {0.0f, 0.5f}, {0.5f, 0.5f}},
      {{0.5f, 0.5f}, {0.5f, 0.0f}, {0.0f, 0.0f}},
      {{1.0f, 0.0f}, {0.5f, 0.0f}, {0.5f, 0.0f}},
      {{0.5f, 0.5f}, {1.0f, 0.5f}, {1.0f, 0.0f}},
      {{1.0f, 1.0f}, {1.0f, 0.5f}, {0.5f, 0.5f}},
      {{0.5f, 0.5f}, {0.5f, 1.0f}, {1.0f, 1.0f}},
      {{0.0f, 1.0f}, {0.5f, 1.0f}, {0.5f, 0.5f}},
      {{0.5f, 0.5f}, {0.0f, 0.5f}, {0.0f, 1.0f}}};

  final static int EDGESWITCH [][] =
    {{0, 0}, {0, 0}, {0, 0}};

  Tile tiles[];

  ProgrammableMatter(int x, int y){
    int i, j, k, l, m;
    float[][][] points = new float [8*x*y][3][2];

    m=0;
    for ( i = 0 ; i< x ; i++) {
      for (j = 0; j < y ; j++) {
        for (k = 0; k < 8 ; k++) {
          for (l = 0; l < 3 ; l++) {

            points[m][l][0] = POINTS[k][l][0] / (float) x + ( (float)1 / (
                  float)x ) * (float)i;
            points[m][l][1] = POINTS[k][l][1] / (float) y + ( (float)1 / (
                  float)y ) * (float)j;


          }
          m++;
        }
      }
    }

    tiles = new Tile[8 * x * y];
    for ( i = 0 ; i < 8 * x * y ; i += 2 ) {
      tiles[i] = new Tile(Tile.SMA, points[i], EDGESWITCH);
      tiles[i + 1] = new Tile(Tile.NO_SMA, points[i+1], EDGESWITCH);
    }
  }

  void setTileSwitch(float pointX0
              , float pointY0
              , float pointX1
              , float pointY1
              , boolean typeOfTile
              , int switchId
              , int typeOfEdge){
    int i;
    int edgeId;

    for (i=0; i<tiles.length; i++) {
      edgeId = tiles[i].hasTwoPoints(pointX0, pointY0, pointX1, pointY1);
      if (tiles[i].getTypeOfTile() == typeOfTile
          && edgeId != -1){
        tiles[i].setEdgeSwitch(typeOfTile, edgeId, switchId, typeOfEdge);
      }
    }
  }

  void printCodeOfTiles(){
    int i, j;
    j = 0;
    for (i = 0; i < tiles.length; i++) {
      System.out.printf("%s \n", tiles[i].getTileCode());
      if (j < 7) {
        j++;
      } else {
        j=0;
        System.out.printf("\n ");
      }
    }
  }

  void run(PlanForOrigami[] planForOrigami) {
    int i, j;
    int numberOfEdges;
    int numberOfPhases;

    for (i = 0; i < 1; i++) {
      int [][] planTable = planForOrigami[i].getPlanTable();
      float [][] edgeTable = planForOrigami[i].getEdgeTable();

      numberOfEdges = planForOrigami[i].getNumberOfEdges();
      numberOfPhases = planForOrigami[i].getNumberOfPhases();

      for(j = 0 ; j < numberOfEdges; j++){
        if (planTable[0][j] == 1){
          setTileSwitch(edgeTable [j][0]
              , edgeTable [j][1]
              , edgeTable [j][2]
              , edgeTable [j][3]
              , Tile.SMA
              , 0
              , 1);

        }
        if (planTable[0][j] == -1){
          setTileSwitch(edgeTable [j][0]
              , edgeTable [j][1]
                , edgeTable [j][2]
              , edgeTable [j][3]
              , Tile.SMA
              , 0
              , 2);


        }
      }
    }
    printCodeOfTiles();
  }
}
```

```
package com.drancom.programmableMatter.tile.clipSMA;

import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.
      MonitorOfPlanGroupOfPlanForOrigamis;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
      .PlanForOrigami;
import com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigamis.PlanForOrigamis;
import com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigamis.PlanerForOrigamis;
import com.drancom.programmableMatter.folding.simulator.
      simulatorForOrigamis.
      SimulatorToFindOptimalOrigamisWithInvertingAndRotation;

public class SimulatorForTileProgrammableMatter {

  public static final String PLAN_FILENAME[] = {
    "c:\\foldingdata\\save_8x8elephant\\plan_for_origami_save_8x8elephant.
        csv"
    , "c:\\foldingdata\\save_8x8sailboat\\plan_for_origami_8x8sailboat.csv"
    , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"
    , "c:\\foldingdata\\save_8x8table\\plan_for_origami_save_8x8table.csv"
    , "c:\\foldingdata\\save_8x8bench\\plan_for_origami_save_8x8bench.csv"
    , "c:\\foldingdata\\save_8x8airplain\\plan_for_origami_8x8airplain.csv"
    , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"

    };

  public static final int NUMBER_OF_PLAN_FILES= 1;

  public static final String PLAN_FOR_ORIGAMIS_FILENAME = "c:\\foldingdata\\save_plan\\%splan_for_origamis %d.%s";
  public static final String PLAN_FOR_ORIGAMIS_FILETYPE = "csv";


  void run() {
    int i;
    int j;
    int k;
/**/
    FileObj[] fileObjs;
    PlanForOrigami[] plansForOrigami;
    PlanForOrigami[] inputPlansForOrigami;
    MonitorOfPlanGroupOfPlanForOrigamis monitor;

    Planner [] planers;

    int numberOfPlansForOrigamis;
    int optimalPlanForOrigamis;
    int numberOfActiveEdgeOfOptimalPlanForOrigamis;

    int optimalGroupsForOrigamis;
    int numberOfGroupsOfOptimalNumberOfGroups;

    String tempString;

    // Initiation
    fileObjs =  new FileObj[NUMBER_OF_PLAN_FILES];
    plansForOrigami = new PlanForOrigami[NUMBER_OF_PLAN_FILES];

    // loads

    for (i=0; i < NUMBER_OF_PLAN_FILES; i++) {
        plansForOrigami[i] = new PlanForOrigami();
        plansForOrigami[i].load(PLAN_FILENAME[i]);
    }

    ProgrammableMatter pm = new ProgrammableMatter(4, 4);
    pm .run(plansForOrigami);

  }

  public static void main(String[] args) {
    SimulatorForTileProgrammableMatter simulator = new SimulatorForTileProgrammableMatter();
    simulator.run();
  }

}
```

```
package com.drancom.programmableMatter.tile.clipSMA;

public class Tile {
  final static boolean SMA  = true;
  final static boolean NO_SMA = false;

  boolean typeOfTile;
  float points[][] = new float[3][2];

  int edgeSwitchs[][] = new int[3][2];

  Tile (boolean typeOfTile, float [][] points, int[][] edgeSwitchs) {
    int i,j;
    this.typeOfTile = typeOfTile;
    for (i=0; i<points.length; i++) {
      for (j=0; j < points[i].length; j++) {
        this.points[i][j] = points[i][j];
      }
    }

    for (i=0; i<edgeSwitchs.length; i++) {
      for (j=0; j < edgeSwitchs[i].length; j++) {
        this.edgeSwitchs[i][j] = edgeSwitchs[i][j];
      }
    }
  }

  boolean isSMA(){
    return (typeOfTile==SMA);
  }

  boolean isNO_SMA(){
    return (typeOfTile==NO_SMA);
  }

  boolean getTypeOfTile() {
    return typeOfTile;
  }

  int hasTwoPoints(float pointX0, float pointY0, float pointX1, float pointY1){
  // return edge enumber

    int i;

    for (i=0; i<3; i++) {
      if ((points[i][0] == pointX0 && points[i][1] == pointY0
          && points[(i+1)%3][0] == pointX1 && points[(i+1)%3][1] == pointY1 )
          ||
        (points[i][0] == pointX1 && points[i][1] == pointY1
          && points[(i+1)%3][0] == pointX0 && points[(i+1)%3][1] == pointY0 )) {
        return i;
      }
    }
    return -1;
  }

  boolean hasPoint(float pointX0, float pointY0) {
    int i;
    for (i=0; i<3; i++) {
      if (points[i][0] == pointX0 && points[i][1] == pointY0) {
        return true;
      }
    }
    return false;
  }

  boolean setEdgeSwitch(boolean typeOfTile
          , int edgeId
          , int switchId
          , int typeOfEdge){
    int i;

    if (this.typeOfTile == typeOfTile && edgeId < 3 && switchId < 2) {
      edgeSwitchs [edgeId][switchId] = typeOfEdge;
      return true;
    }

    return false;
  }
  String getTileCode () {

    String power0 = new String();
    String power1 = new String();
    int code0 = 0;
    int code1 = 0;
    power0 += Integer.toString(edgeSwitchs[0][0]);
    power0 += Integer.toString(edgeSwitchs[1][0]);
    power0 += Integer.toString(edgeSwitchs[2][0]);

    power1 += Integer.toString(edgeSwitchs[0][1]);
    power1 += Integer.toString(edgeSwitchs[1][1]);
    power1 += Integer.toString(edgeSwitchs[2][1]);

    if (edgeSwitchs[0][0]!=0) {
      code0 += 1;
    }
    if (edgeSwitchs[1][0]!=0) {
      code0 += 2;
    }
    if (edgeSwitchs[2][0]!=0) {
      code0 += 4;
    }

    if (edgeSwitchs[0][1]!=0) {
      code1 += 1;
    }
    if (edgeSwitchs[1][1]!=0) {
      code1 += 2;
    }
    if (edgeSwitchs[2][1]!=0) {
      code1 += 4;
    }

    return power0 + ” ” + power1 + ” ” + Integer.toString(code0) + Integer.
        toString(code1) ;
  }
}
```

```
package com.drancom.programmableMatter.tile.springSma;

import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
    .PlanForOrigami;

public class ProgrammableMatter {
  final static float POINTS[][][] =
      {{{0.0f, 0.0f}, {0.0f, 0.5f}, {0.5f, 0.5f}},
      {{0.5f, 0.5f}, {0.5f, 0.0f}, {0.0f, 0.0f}},
      {{1.0f, 0.0f}, {0.5f, 0.0f}, {0.5f, 0.0f}},
      {{0.5f, 0.5f}, {1.0f, 0.5f}, {1.0f, 0.0f}},
      {{1.0f, 1.0f}, {1.0f, 0.5f}, {0.5f, 0.5f}},
      {{0.5f, 0.5f}, {0.5f, 1.0f}, {1.0f, 1.0f}},
      {{0.0f, 1.0f}, {0.5f, 1.0f}, {0.5f, 0.5f}},
      {{0.5f, 0.5f}, {0.0f, 0.5f}, {0.0f, 1.0f}}};

  final static boolean EDGESWITCH [][] =
    {{false, false}, {false, false}, {false, false}};

  Tile tiles[];

  ProgrammableMatter(int x, int y){
    int i, j, k, l, m;
    float[][][] points = new float [8*x*y][3][2];

    m=0;
    for ( i = 0 ; i< x ; i++) {
      for (j = 0; j < y ; j++) {
        for (k = 0; k < 8 ; k++) {
          for (l = 0; l < 3 ; l++) {

            points[m][l][0] = POINTS[k][l][0] / (float) x + ( (float)1 / (
                float)x ) * (float)i;
            points[m][l][1] = POINTS[k][l][1] / (float) y + ( (float)1 / (
                float)y ) * (float)j;


          }
          m++;
        }
      }
    }

    tiles = new Tile[8 * x * y];
    for ( i = 0 ; i < 8 * x * y ; i += 2 ) {
      tiles[i] = new Tile(Tile.MOUNT, points[i], EDGESWITCH);
      tiles[i + 1] = new Tile(Tile.VALLEY, points[i+1], EDGESWITCH);
    }
  }

  void setTileSwitch(float pointX0
            , float pointY0
            , float pointX1
            , float pointY1
            , boolean typeOfTile
            , int switchId
            , boolean isTurnOn){
    int i;
    int edgeId;

    for (i=0; i<tiles.length; i++) {
      edgeId = tiles[i].hasTwoPoints(pointX0, pointY0, pointX1, pointY1);
      if (tiles[i].getTypeOfTile() == typeOfTile
          && edgeId != -1){
        tiles[i].setEdgeSwitch(typeOfTile, edgeId, switchId, isTurnOn);
      }
    }
  }

  void printCodeOfTiles(){
    int i, j;
    j = 0;
    for (i = 0; i < tiles.length; i++) {
      System.out.printf(”%d ”, tiles[i].getTileCode());
      if (j < 7) {
        j++;
      } else {
        j=0;
        System.out.printf(”\n ”);
      }
    }
  }

  void run(PlanForOrigami[] planForOrigami) {
    int i, j;
    int numberOfEdges;
    int numberOfPhases;

    for (i = 0; i < 1; i++) {
      int [][] planTable = planForOrigami[i].getPlanTable();
      float [][] edgeTable = planForOrigami[i].getEdgeTable();

      numberOfEdges = planForOrigami[i].getNumberOfEdges();
      numberOfPhases = planForOrigami[i].getNumberOfPhases();

      for(j = 0 ; j < numberOfEdges; j++){
        if (planTable[0][j] == 1){
          setTileSwitch(edgeTable [j][0]
              , edgeTable [j][1]
              , edgeTable [j][2]
```

```
                    , edgeTable [j][3]
                    , Tile.MOUNT
                    , 0
                    , true);

          }
          if (planTable[0][j] == -1){
             setTileSwitch(edgeTable [j][0]
                    , edgeTable [j][1]
                      , edgeTable [j][2]
                    , edgeTable [j][3]
                    , Tile.VALLEY
                    , 0
                    , true);


          }
        }
      }
      printCodeOfTiles();
   }
}
```

```
package com.drancom.programmableMatter.tile.springSma;

import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.
       MonitorOfPlanGroupOfPlanForOrigamis;
import com.drancom.programmableMatter.folding.origami.planner.Planner;
import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
       .PlanForOrigami;
import com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamis.PlanForOrigamis;
import com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamis.PlanerForOrigamis;
import com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamis.
       SimulatorToFindOptimalOrigamisWithInvertingAndRotation;

public class SimulatorForTileProgrammableMatter {

  public static final String PLAN_FILENAME[] = {
    "c:\\foldingdata\\save_8x8elephant\\plan_for_origami_save_8x8elephant.
         csv"
    , "c:\\foldingdata\\save_8x8sailboat\\plan_for_origami_8x8sailboat.csv"
    , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"
    , "c:\\foldingdata\\save_8x8table\\plan_for_origami_save_8x8table.csv"
    , "c:\\foldingdata\\save_8x8bench\\plan_for_origami_save_8x8bench.csv"
    , "c:\\foldingdata\\save_8x8airplain\\plan_for_origami_8x8airplain.csv"
    , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"

    };

  public static final int NUMBER_OF_PLAN_FILES= 1;

  public static final String PLAN_FOR_ORIGAMIS_FILENAME = "c:\\foldingdata
        \\save_plan\\%splan_for_origamis %d.%s";
  public static final String PLAN_FOR_ORIGAMIS_FILETYPE = "csv";


  void run() {
    int i;
    int j;
    int k;
/**/
    FileObj[] fileObjs;
    PlanForOrigami[] plansForOrigami;
    PlanForOrigami[] inputPlansForOrigami;
    MonitorOfPlanGroupOfPlanForOrigamis monitor;

    Planner [] planers;

    int numberOfPlansForOrigamis;
    int optimalPlanForOrigamis;
    int numberOfActiveEdgeOfOptimalPlanForOrigamis;

    int optimalGroupsForOrigamis;
    int numberOfGroupsOfOptimalNumberOfGroups;

    String tempString;

    // Initiation
    fileObjs = new FileObj[NUMBER_OF_PLAN_FILES];
    plansForOrigami = new PlanForOrigami[NUMBER_OF_PLAN_FILES];

    // loads

    for (i=0; i < NUMBER_OF_PLAN_FILES; i++) {
        plansForOrigami[i] = new PlanForOrigami();
        plansForOrigami[i].load(PLAN_FILENAME[i]);
    }

    ProgrammableMatter pm = new ProgrammableMatter(4, 4);
    pm .run(plansForOrigami);

  }

  public static void main(String[] args) {
    SimulatorForTileProgrammableMatter simulator = new
          SimulatorForTileProgrammableMatter();
    simulator.run();
  }

}
```

```
package com.drancom.programmableMatter.tile.springSma;

public class Tile {
  final static boolean MOUNT  = true;
  final static boolean VALLEY = false;

  boolean typeOfTile;
  float points[][] = new float[3][2];

  boolean edgeSwitchs[][] = new boolean[3][2];

  Tile (boolean typeOfTile, float [][] points, boolean[][] edgeSwitchs) {
    int i,j;
    this.typeOfTile = typeOfTile;
    for (i=0; i<points.length; i++) {
      for (j=0; j < points[i].length; j++) {
        this.points[i][j] = points[i][j];
      }
    }

    for (i=0; i<edgeSwitchs.length; i++) {
      for (j=0; j < edgeSwitchs[i].length; j++) {
        this.edgeSwitchs[i][j] = edgeSwitchs[i][j];
      }
    }

  }

  boolean isMount(){
    return (typeOfTile==MOUNT);
  }

  boolean isValley(){
    return (typeOfTile==VALLEY);
  }

  boolean getTypeOfTile() {
    return typeOfTile;
  }

  int hasTwoPoints(float pointX0, float pointY0, float pointX1, float
       pointY1){
  // return edge enumber

    int i;

    for (i=0; i<3; i++) {
      if ((points[i][0] == pointX0 && points[i][1] == pointY0
         && points[(i+1)%3][0] == pointX1 && points[(i+1)%3][1] == pointY1
              )
          ||
        (points[i][0] == pointX1 && points[i][1] == pointY1
         && points[(i+1)%3][0] == pointX0 && points[(i+1)%3][1] == pointY0
              )) {
        return i;
      }
    }
    return -1;
  }

  boolean hasPoint(float pointX0, float pointY0) {
    int i;
    for (i=0; i<3; i++) {
      if (points[i][0] == pointX0 && points[i][1] == pointY0) {
        return true;
      }
    }
    return false;
  }

  boolean setEdgeSwitch(boolean typeOfTile
          , int edgeId
          , int switchId
          , boolean isTurnOn){
    int i;

    if (this.typeOfTile == typeOfTile && edgeId < 3 && switchId < 2) {
      edgeSwitchs [edgeId][switchId] = isTurnOn;
      return true;
    }

    return false;
  }
  int getTileCode () {
    int powerl, power0;
    powerl = 0;
    power0 = 0;
    if (edgeSwitchs[0][0]) {
      power0 += 1;
    }
    if (edgeSwitchs[1][0]) {
      power0 += 2;
    }
    if (edgeSwitchs[2][0]) {
      power0 += 4;
    }
    if (edgeSwitchs[0][1]) {
      powerl += 1;
    }
    if (edgeSwitchs[1][1]) {
      powerl += 2;
    }
    if (edgeSwitchs[2][1]) {
      powerl += 4;
    }
    return powerl * 10 + power0;
  }
}
```

```
package com.drancom.programmableMatter.unfolding.unfolder;

import java.util.ArrayList;

import com.drancom.programmableMatter.folding.controller.paper.Paper;

public interface Unfolder {
  public ArrayList<Paper> unfolding(Paper paper);
}
```

```
package com.drancom.programmableMatter.unfolding.energyFunction;

import com.drancom.programmableMatter.folding.controller.paper.Line;
import com.drancom.programmableMatter.folding.controller.paper.Paper;

public class Unfolder {
  Paper foldedPapers[];
  int numberOfLine;
  int numberOfPoint;

  public Paper[] unfolding (Paper paper, int numberOfPapers) {
    Line line;
    Paper oldPaper;
    Paper alternatePaper;
    Paper foldedPaper[] = new Paper[numberOfPapers];

    // initiation


    foldedPapers = new Paper [numberOfPapers];

    numberOfLine = paper.getNumberOfEdges();
    numberOfPoint = paper.getNumberOfEdges();

    int i, j;

    //
    for (i=0; i<numberOfPapers; i++) {
      for (j=0; j<numberOfLine; j++){
        line = paper.getLine(j);
        if (line.getAngle() != 0.0f) {
          // change the angle + or -

        }


      }
    }
    return foldedPaper;
  }


}
```

```
package com.drancom.programmableMatter.unfolding.energyFunction;

import java.util.ArrayList;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.util.
      Mathematica;
import com.drancom.programmableMatter.folding.dataFile.FileObj;
import com.drancom.programmableMatter.folding.monitor.MainWindow;
import com.drancom.programmableMatter.folding.monitor.MonitorOfPaperArray;
import com.drancom.programmableMatter.unfolding.globalEnergyFunction.
      UnfolderWithGlobalEnergyFunction;

public class UnfoldingByEnergyFunc {
  /**/
  public static final String FILENAME = "c:\\foldingdata\\save_box\\4
      x4box_folded.obj";
  /** /
  public static final String FILENAME = "c:\\foldingdata\\save_2x2mountain
      \\2x2mountain_folded.obj";
  /**/
  Paper paper;

  Paper[] papers;
  FileObj fileObj;
  MonitorOfPaperArray monitorOfPaperArray;

  void run() {
    int i;
    String fileName;
    ArrayList<Paper> paperArray;
    ArrayList<Paper> reversPaperArray = new ArrayList<Paper>();

    // init
    paper = new Paper();
    fileObj    = new FileObj();

    UnfolderWithGlobalEnergyFunction unfolder = new
          UnfolderWithGlobalEnergyFunction ();

    monitorOfPaperArray = new MonitorOfPaperArray();

    // load
    fileName = String.format(FILENAME);
    fileObj.load(fileName, paper);

    Mathematica mathematica = new Mathematica();
    mathematica.load();


    /**/
    paperArray = unfolder.unfolding(paper);
    for (i = paperArray.size() ; i>= 0 ; i--){
      reversPaperArray.add( paperArray.get(i));
    }

    /** /
    paperArray = new ArrayList<Paper>();
    paperArray.add(paper);
    /**/
    mathematica.close();
    monitorOfPaperArray.run(paperArray);
  }

  public static void main(String[] args) {
    UnfoldingByEnergyFunc unfoldingByEnergyFunc = new UnfoldingByEnergyFunc
        ();
    unfoldingByEnergyFunc.run();
  }
}
```

```
package com.drancom.programmableMatter.unfolding.globalEnergyFunction;

import java.util.ArrayList;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.Point;
import com.drancom.programmableMatter.folding.controller.paper.Polygon;
import com.drancom.programmableMatter.folding.monitor.MonitorOfPaperArray;
import com.drancom.programmableMatter.unfolding.unfolder.Unfolder;

public class UnfolderWithGlobalEnergyFunction implements Unfolder{

  @Override
  public ArrayList<Paper> unfolding(Paper paper) {
    /**
     *
     * 1. try unfolding all of the lines has edges Type A: Unfolding each
     * line as much as possible. Type B: Unfolding each line until E became
     * the smallest. 2. pick the paper has smallest energy 3. record array.
     * 4. repeat 1 and 2 until all of the lines are unfolded.
     */
    int i, j, k;

    ArrayList<Paper> paperArray = new ArrayList<Paper>();

    Paper minEngPaper;
    Paper unfoldingPaper;
    Paper workingPaper;
    Polygon polygon;
    Point[] points;

    // minEngPaper
    unfoldingPaper = paper.snapshot();
    minEngPaper = unfoldingPaper.snapshot();

    paperArray.add(unfoldingPaper.snapshot());

    for (k = 0; k < paper.getNumberOfPolygons(); k++) {
      for (i = 0; i < paper.getNumberOfPolygons(); i++) {
        for (j = 0; j < 3; j++) {
          workingPaper = unfoldingPaper.snapshot();
          polygon = workingPaper.getPolygon(i);
          points = polygon.getPoints();

          if (workingPaper.changeAngleAsMuchAsPossible(polygon, points[j],
              points[(j + 1) % 3])) {

            // compare minEnergy Paper and paper
            if (minEngPaper.getGlobalEnergy() > workingPaper
                .getGlobalEnergy()) {
              minEngPaper = workingPaper;
            }

            // unfolded
            if (workingPaper.getGlobalEnergy() == 0.0f) {
              paperArray.add(minEngPaper);
              if (minEngPaper.getGlobalEnergy() == 0.0f) {
                paperArray.add(0, paper.snapshot());
                return paperArray;
              }
            }
          }
        }
      }

      unfoldingPaper = minEngPaper;
      paperArray.add(minEngPaper);
      System.out.println(minEngPaper.getGlobalEnergy());
/**/
      if (minEngPaper.getGlobalEnergy() <= Paper.ERROR_RATIO_FOR_LANGTH ) {
        MonitorOfPaperArray monitor = new MonitorOfPaperArray();
        monitor.run(paperArray);
        }
/**/
      if (minEngPaper.getGlobalEnergy()== 0
           || (paperArray.size()>3 && (paperArray.get(paperArray.size()-1).
               getGlobalEnergy()
          == paperArray.get(paperArray.size()-3).getGlobalEnergy() ))) {
        break;
      }
    }

    for(i=0 ; i < paperArray.size(); i++) {
      System.out.println(paperArray.get(i).getGlobalEnergy());
    }

    return paperArray;
  }
}
```

```
package com.drancom.programmableMatter.unfolding.localEnergyFunction;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Hashtable;
import java.util.SortedMap;

import com.drancom.programmableMatter.folding.controller.paper.Paper;
import com.drancom.programmableMatter.folding.controller.paper.Point;
import com.drancom.programmableMatter.folding.controller.paper.Polygon;
import com.drancom.programmableMatter.folding.monitor.MonitorOfPaperArray;
import com.drancom.programmableMatter.unfolding.unfolder.Unfolder;

public class UnfolderWithLocalEnergyFunction implements Unfolder{


  @Override
  public ArrayList<Paper> unfolding(Paper paper) {
    /**
     *
     * 1. try unfolding all of the lines has edges Type A: Unfolding each
     * line as much as possible. Type B: Unfolding each line until E became
     * the smallest. 2. pick the paper has smallest energy 3. record array.
     * 4. repeat 1 and 2 until all of the lines are unfolded.
     */
    int i, j, k;

    ArrayList<Paper> paperArray = new ArrayList<Paper>();

    Paper minEngPaper;
    Paper unfoldingPaper;
    Paper workingPaper;
    Polygon polygon;
    Point[] points;

    float [] listOflocalEnergy       = new float[paper.getNumberOfEdges()];
    float [] sortedListOflocalEnergy  = new float[paper.getNumberOfEdges()
          ];
    int   [] listOfIndexOfLocalEnergy = new int[paper.getNumberOfEdges()];

    // minEngPaper
    unfoldingPaper = paper.snapshot();
    minEngPaper = unfoldingPaper.snapshot();

    paperArray.add(unfoldingPaper.snapshot());



    for (k = 0; k < paper.getNumberOfEdges(); k++) {

      // get a list of lines sorted by energy function

      workingPaper = unfoldingPaper.snapshot();

      for (i=0; i<paper.getNumberOfEdges(); i++) {

        listOflocalEnergy[i]     = workingPaper.getLocalEnergy(workingPaper.
             getLine(i));
        sortedListOflocalEnergy[i]  = listOflocalEnergy[i];
      }
      Arrays.sort(sortedListOflocalEnergy);
      for (i=paper.getNumberOfEdges() - 1; i <= 0; i--) {
        for (j=0; j<=paper.getNumberOfEdges() ; j++ ) {
          if (sortedListOflocalEnergy[i] == listOflocalEnergy[j]) {
            listOfIndexOfLocalEnergy[(paper.getNumberOfEdges() - 1) - i] =
                j;
            break;
          }
        }
      }

      for (i = 0; i<workingPaper.getNumberOfEdges(); i++ ) {
        if (workingPaper.changeAngleForSmallestGlobalEnergy(
          workingPaper.getLine(listOfIndexOfLocalEnergy[i]))) {

        // compare minEnergy Paper and paper
        if (minEngPaper.getGlobalEnergy() > workingPaper
          .getGlobalEnergy()) {
          minEngPaper = workingPaper;
        }
            // unfolded
          if (workingPaper.getGlobalEnergy() == 0.0f) {
            paperArray.add(minEngPaper);
            if (minEngPaper.getGlobalEnergy() == 0.0f) {
              paperArray.add(0, paper.snapshot());
              return paperArray;
            }
          }
        }
      }

      unfoldingPaper = minEngPaper;
      paperArray.add(minEngPaper);
      System.out.println(minEngPaper.getGlobalEnergy());
/**/
      if (minEngPaper.getGlobalEnergy() <= Paper.ERROR_RATIO_FOR_LANGTH ) {
        MonitorOfPaperArray monitor = new MonitorOfPaperArray();
        monitor.run(paperArray);
      }
/**/
      if (minEngPaper.getGlobalEnergy()== 0
          || (paperArray.size()>3 && (paperArray.get(paperArray.size()-1).
               getGlobalEnergy()
           == paperArray.get(paperArray.size()-3).getGlobalEnergy() ))) {
        break;
      }


      for(i=0 ; i < paperArray.size(); i++) {
        System.out.println(paperArray.get(i).getGlobalEnergy());
      }

      return paperArray;
    }
    return paperArray;

  }

}
```

Bibliography

[1] Byoungkwon An. Em-cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3149–3155, May 2008.

[2] Byoungkwon An, Nadia Benbernou, Erik D. Demaine, and Daniela Rus. Planning to fold multiple objects from a single self-folding sheet. *Robotica*, 2011.

[3] Byoungkwon An and Daniela Rus. Making shapes from modules by magnification. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[4] Devin Balkcom. *Robotic Origami Folding.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2004.

[5] Devin Balkcom and Matthew Mason. Robotic origami folding. *International Journal of Robotics Research*, 27(5):613 - 627, May 2008.

[6] Devin J. Balkcom and Matthew T. Mason. Introducing robotic origami folding. In *IEEE International Conference on Robotics and Automation*, pages 3245–3250, 2004.

[7] Nadia Benbernou, Erik D. Demaine, Martin L. Demaine, and Aviv Ovadya. A universal crease pattern for folding orthogonal shapes.

[8] Zack Butler, Keith Kotay, Daniela Rus, and Koji Tomita. Generic decentralized control for a class of self-reconfigurable robots. In *Proceedings of ICRA*, 2002.

[9] Zack J. Butler and Daniela Rus. Distributed planning and control for modular robots with unit-compressible modules. *I. J. Robotic Res.*, 22(9):699–716, 2003.

[10] G.S. Chirikjian. Kinematics of a metamorphic robotic system. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 449 –455 vol.1, May 1994.

[11] J. S. Dai and D. G. Caldwell. Origami-based robotic paper-and-board packaging for food industry. *Trends in Food Science & Technology*, 21(3):153–157, March 2010.

[12] 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 (CCCG'04)*, pages 64–67, 2004.

[13] Kyle Gilpin, Keith Kotay, Daniela Rus, and Iuliu Vasilescu. Miche: Modular shape formation by self-disassembly. *Int. J. Rob. Res.*, 27(3-4):345–372, 2008.

[14] 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*, 107(28):12441–12445, 2010.

[15] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, Leuven, Belgium, 1998.

[16] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 2858 –2863 vol.4, May 1998.

[17] Eric Joisel. Aragon. WebSite, http://www.ericjoisel.com/gallery.html.

[18] Eric Joisel. Gundalf. WebSite, http://www.ericjoisel.com/gallery.html.

[19] Eric Joisel. Legolas. WebSite, http://www.ericjoisel.com/gallery.html.

[20] Satoshi Kamiya. Ryujin 3.5. WebSite, http://www.folders.jp/g/2005/0501.html.

[21] Michihiko Koseki, Kengo Minami, and Norio Inou. Cellular robots forming a mechanical structure (evaluation of structural formation and hardware design of "chobie ii"). In *Proceedings of 7th International Symposium on Distributed Autonomous Robotic Systems (DARS04)*, pages 131–140, June 2004.

[22] K. Kotay and D. Rus. Efficient locomotion for a self-reconfiguring robot. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2963 – 2969, 2005.

[23] K. Kotay, D. Rus, M. Vona, and K. McGray. The self-reconfiguring robotic molecule: Design and control algorithms. In *In Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, pages 375 – 386, Houston, USA, 1998.

[24] Keith Kotay. *Self-Reconfiguring Robots: Designs, Algorithms, and Applications.* PhD thesis, Dartmouth College, Computer Science Department, 2003.

[25] Keith Kotay and Daniela Rus. Motion synthesis for the self-reconfiguring robotic molecule. In *Proceedings of the 1998 Conference on Intelligent Robot Systems*, 1998.

[26] Keith Kotay and Daniela Rus. Algorithms for self-reconfiguring molecule motion planning. In *IROS*, 2000.

[27] Keith Kotay and Daniela Rus. Locomotion versatility through self-reconfiguration. *In Robots and Autonomous Systems*, 2000.

[28] Keith Kotay, Daniela Rus, and Marsette Vona. Using modular self-reconfiguring robots for locomotion. In *in 7th International Sysmposium on Experimental Robotics*, pages 259–269, 2000.

[29] Keith Kotay, Daniela Rus, Marsette Vona, and Craig McGray. The self-reconfiguring robotic molecule. In *Proceedings of the IEEE Intl. Conference on Robotics and Automation*, volume 1, pages 424–431, Leuven, Belgium, 1998.

[30] Liang Lu and Srinivas Akella. Folding cartons with fixtures: A motion planning approach. *IEEE Transactions on Robotics and Automation*, 16(4):346–356, 2000.

[31] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *in International Conference on Robotics and Automation*, pages 441–448, San Diego, California, May 1994.

[32] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 441 –448 vol.1, May 1994.

[33] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. A 3-d self-reconfigurable structure. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 1, pages 432 –439 vol.1, May 1998.

[34] A. Pamecha, C-J. Chiang, D. Stein, and G. Chirikjian. Design and implementation of metamorphic robots. In *Proc. of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference*, 1996.

[35] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with unit-compressible modules. *Autonomous Robots*, 10(1):107–24, 2001.

[36] John W. Suh, Samuel B. Homans, and Mark Yim. Telecubes: Mechanical design of a module for self-reconfigurable robotics. In *In Proc. of IEEE ICRA*, pages 4095–4101, 2002.

[37] C. Ünsal, H. Kiliççöte, and P. Khosla. A modular self-reconfigurable bipartite robotic system: Implementation and motion planning. *Autonomous Robots*, 10(1):23–40, 2001.

[38] Paul White, Victor Zykov, Josh Bongard, and Hod Lipson. Three dimensional stochastic reconfiguration of modular robots. In *Robotics Science and Systems*. MIT, June 8-10 2005.

[39] R.J. Wood. The first takeoff of a biologically inspired at-scale robotic insect. *Robotics, IEEE Transactions on*, 24(2):341 –347, 2008.

[40] Eiichi Yoshida, Shigeru Kokaji, Satoshi Murata, Kohji Tomita, and Haruhisa Kurokawa. Micro self-reconfigurable robot using shape memory alloy. *Journal of Robotics and Mechatronics*, 13(2):212–219, 2001.