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

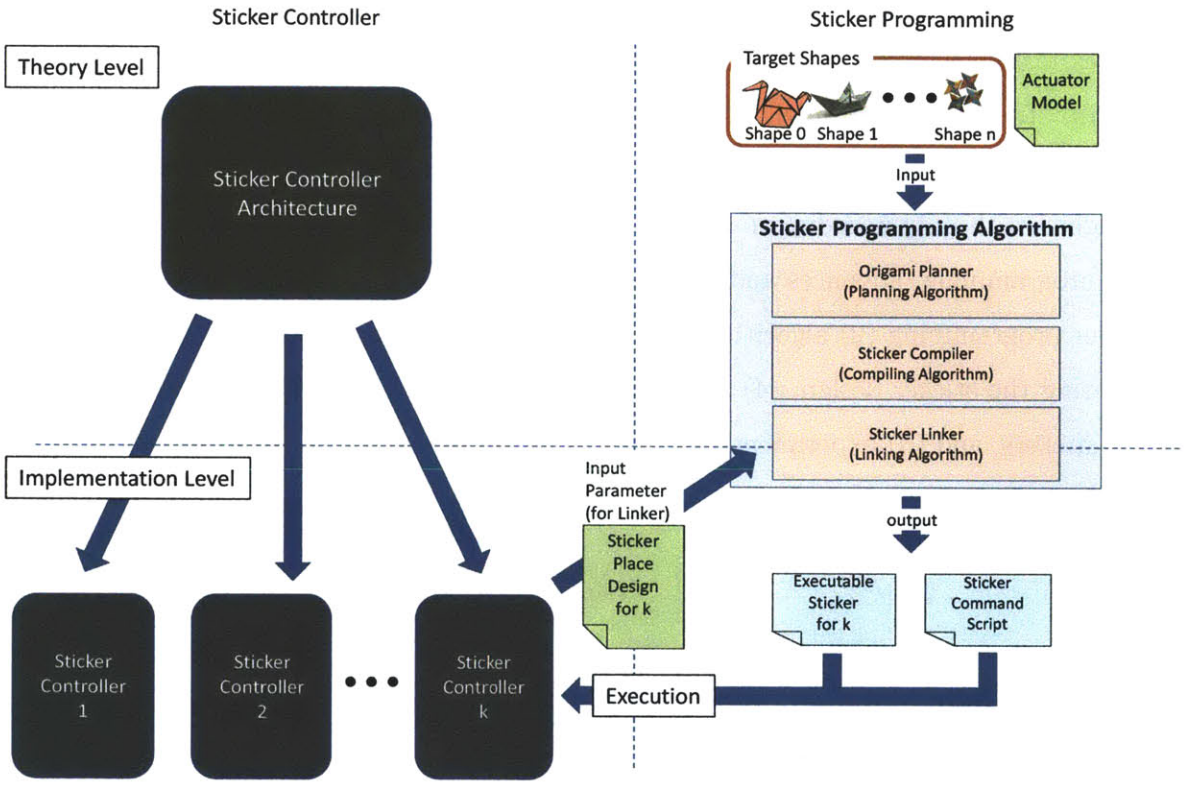


Figure 1-2: Overview of the approach for programming and controlling the self-folding sheet: the sticker controller and the sticker programming.

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.

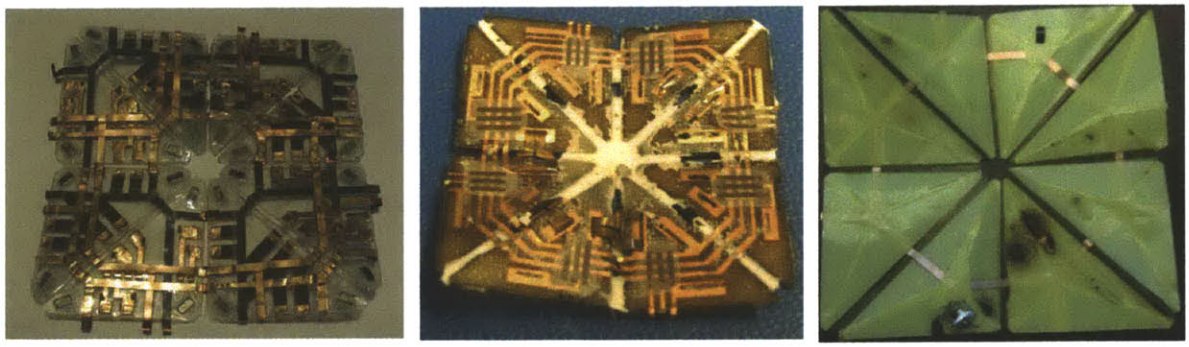


Figure 1-3: Three examples of manufactured 2×2 self-folding sheets. Each device was created using the same architecture but a different manufacturing process

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 $\times$ 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

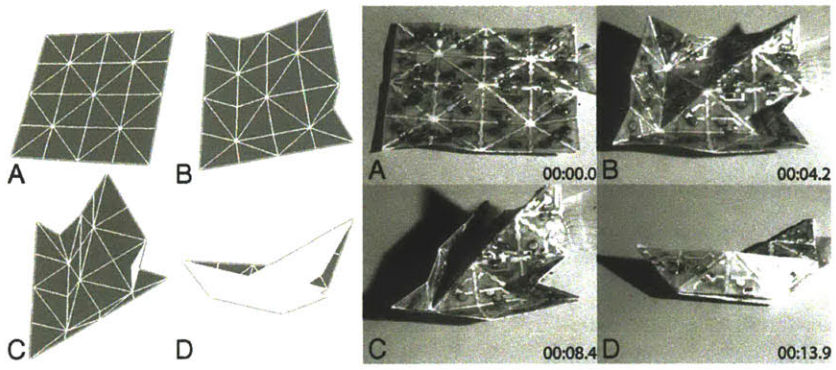


Figure 2-1: Simulation (left) and experiments (right, with time shown in lower right) of a self-folding boat (A). All actuators receiving current (B). Immediately before magnetic closures engage (C). Finished boat on side (D). [14]

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

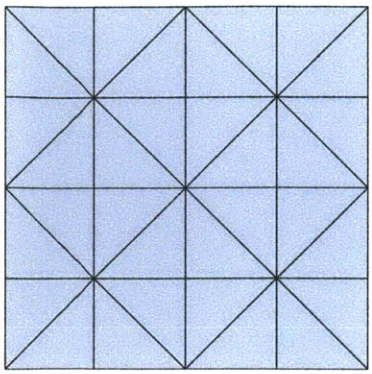


Figure 2-2: Box-pleated crease pattern

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 $\times$ 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 $\pm$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

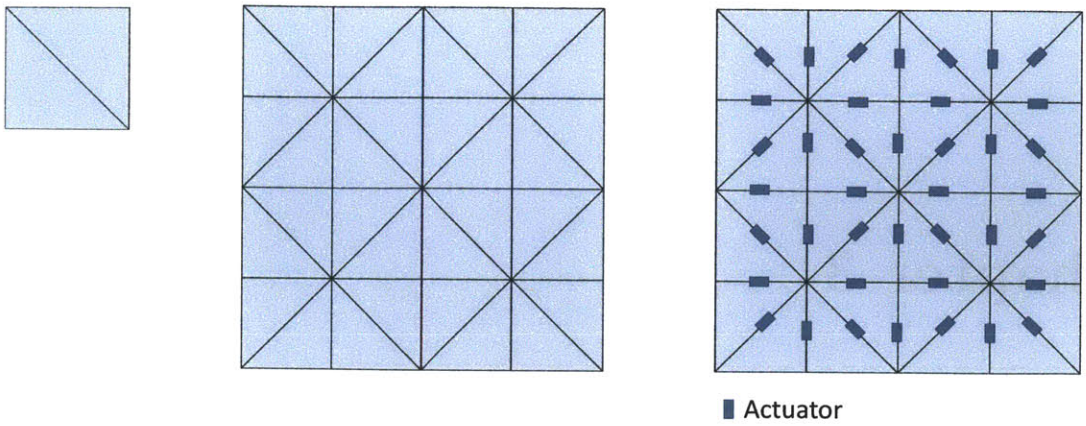


Figure 3-1: An 1×1 self-folding sheet (left) is a basic unit of self-folding sheets. A 4×4 self-folding sheet (middle), and with folding actuators (right). The 4×4 sheet is composed of 16 $(= 4 \times 4)$ 1×1 self-folding sheets

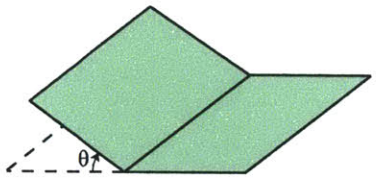


Figure 3-2: The fold angle at a crease is the supplement of the dihedral angle.[2]

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.

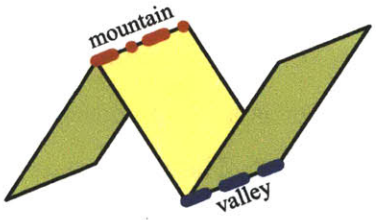


Figure 3-3: A crease can be folded as either a mountain fold (left) or a valley fold (right).[2]

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° }
δ(0000) = 0°
δ(0001) = +90°
δ(0010) = -90°
δ(0100) = +180°
δ(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°

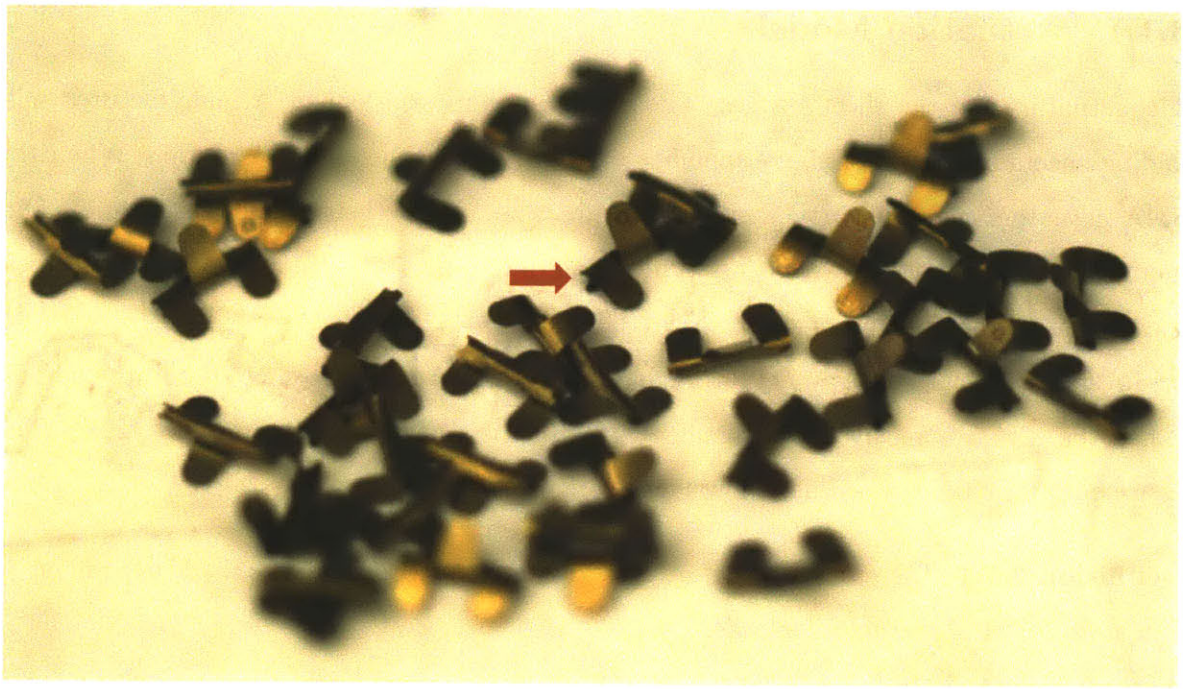


Figure 3-4: Y-Type Actuators. It is an one-directional one-angle actuator. The red arrow points a loop of an actuator.

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):

Σ = { 0, 1 }

A is { 0°, ±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

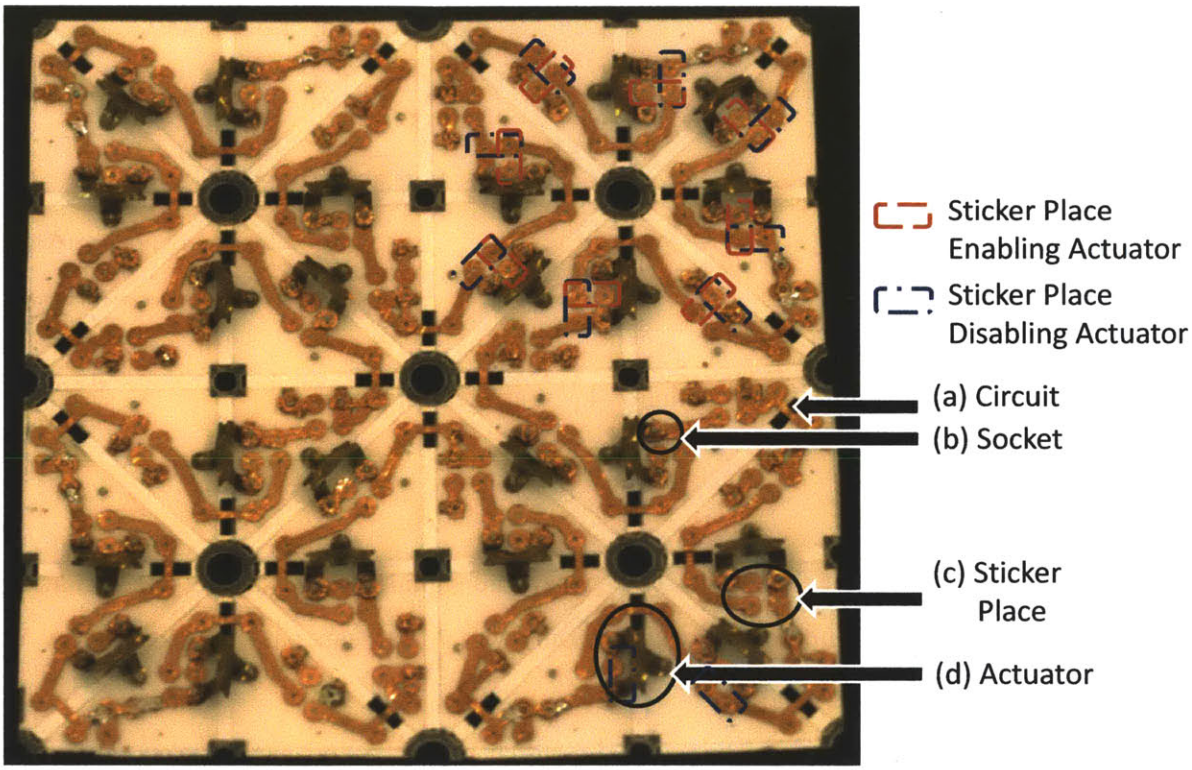


Figure 4-2: The 4 × 4 sticker controller for the self-folding sheet experiment described in Chapter 6.

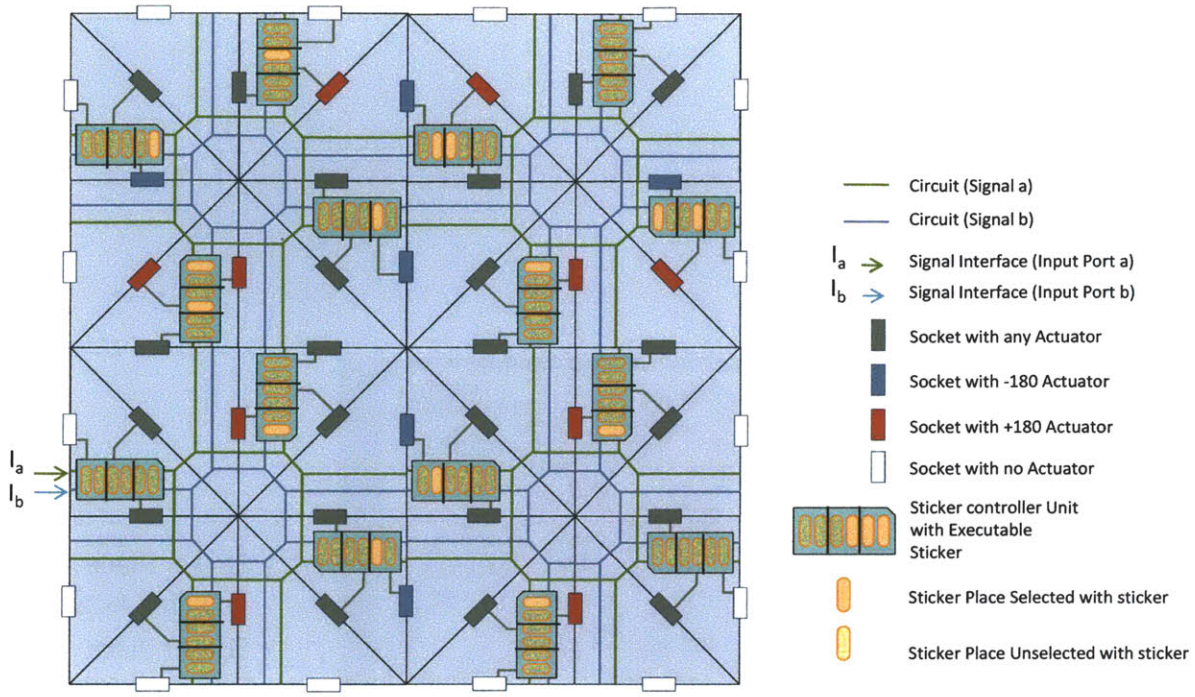


Figure 4-3: Model of 4×4 sticker controller

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.

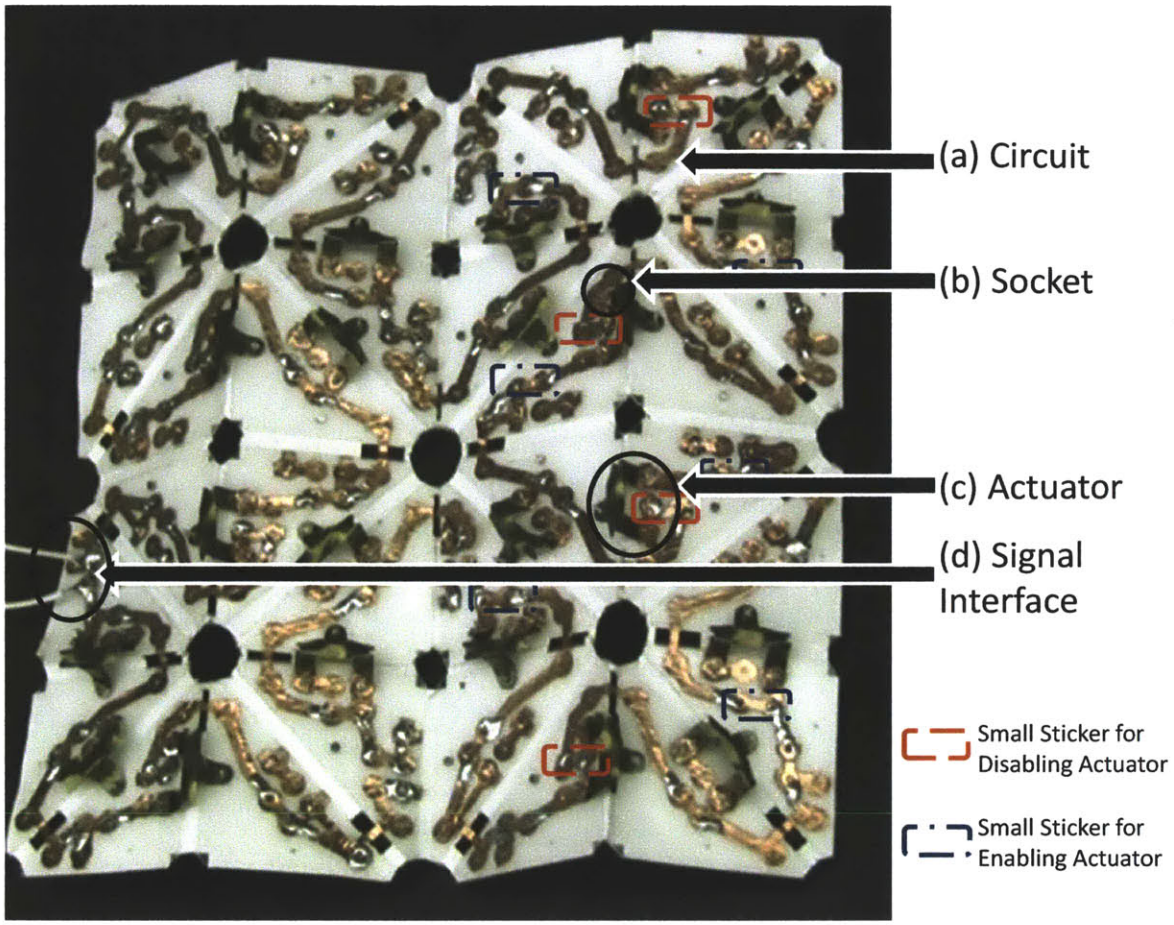


Figure 4-4: 4×4 Self-folding sheet with executable sticker (Vertical folding program).

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.

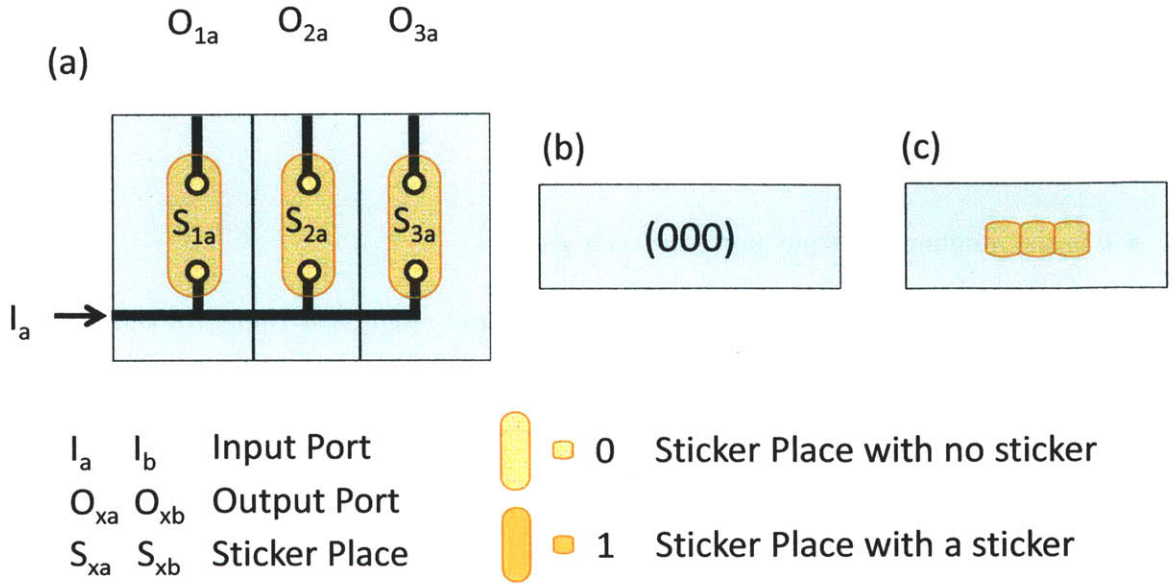


Figure 4-5: Three diagrams for the 3-1-1 sticker controller unit with no sticker. A small arrow mark on the left is the input port. (a), (b), and (c) represent the same sticker controller unit. (a) is the model diagram for the controller unit. (b) is the code diagram for the controller unit. (c) is the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

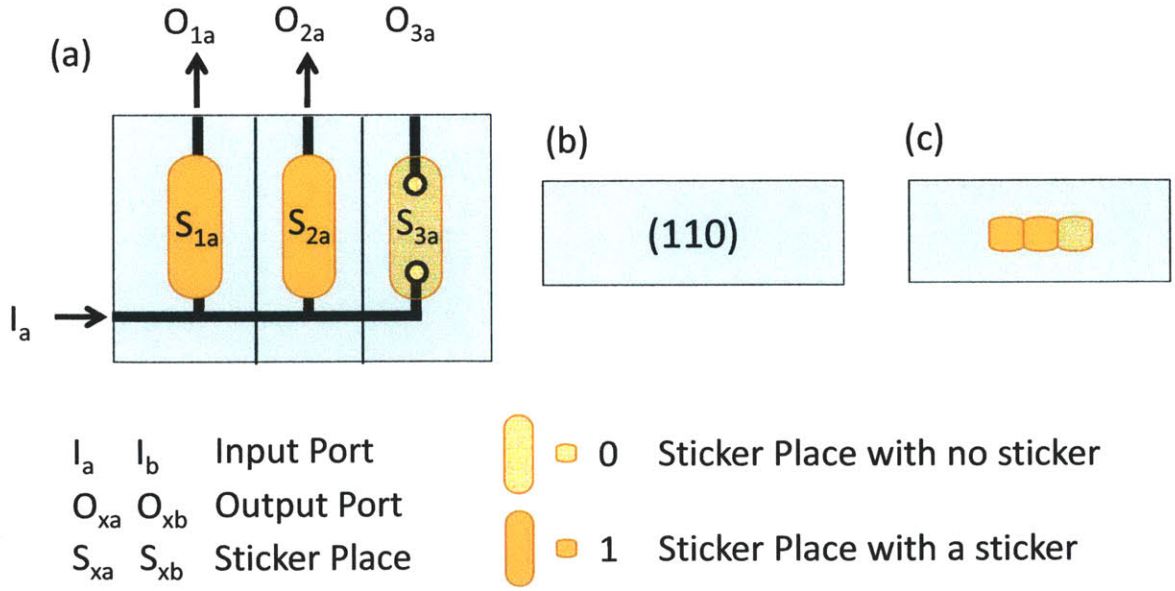


Figure 4-6: Three diagrams for the 3-1-1 sticker controller unit with the stickers. (a), (b), and (c) represent the same sticker controller unit. (a) is the model diagram for the controller unit. (b) is the code diagram for the controller unit. (c) is the sticker diagram for the controller unit.

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.

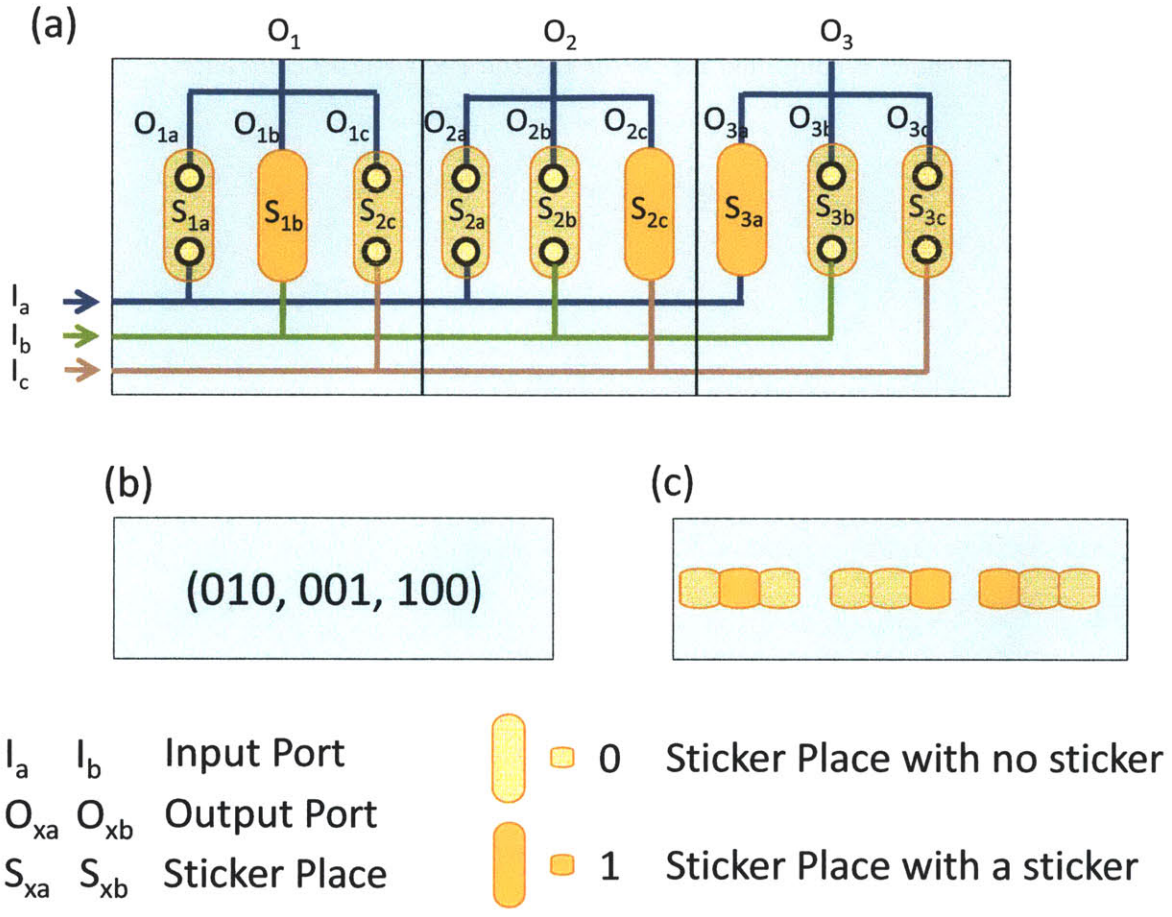


Figure 4-7: Three diagrams for the 3-3-1 sticker controller unit with stickers. (a), (b), and (c) represent the same sticker controller unit. (a) is the model diagram for the controller unit. (b) is the code diagram for the controller unit. (c) is the sticker diagram for the controller unit.

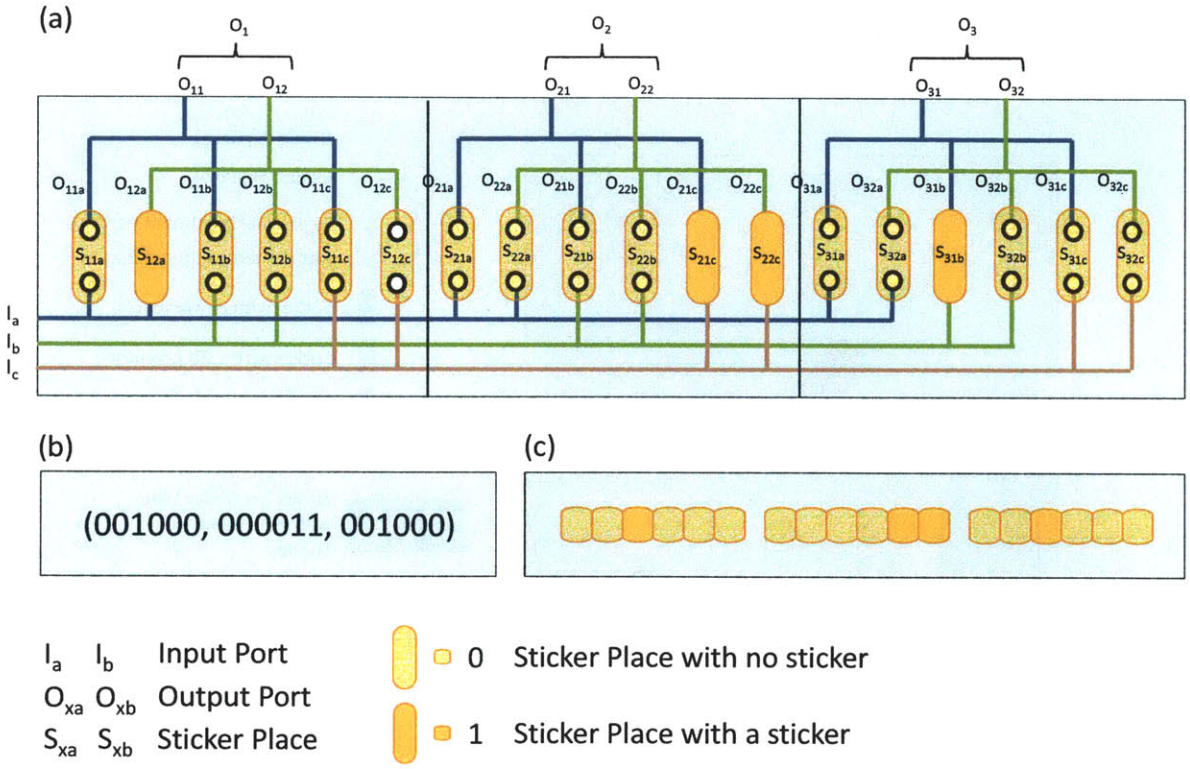


Figure 4-8: Three diagrams for the 3-3-2 sticker controller unit with stickers. (a), (b), and (c) represent the same sticker controller unit. (a) is the model diagram for the controller unit. (b) is the code diagram for the controller unit. (c) is the sticker diagram for the controller unit.

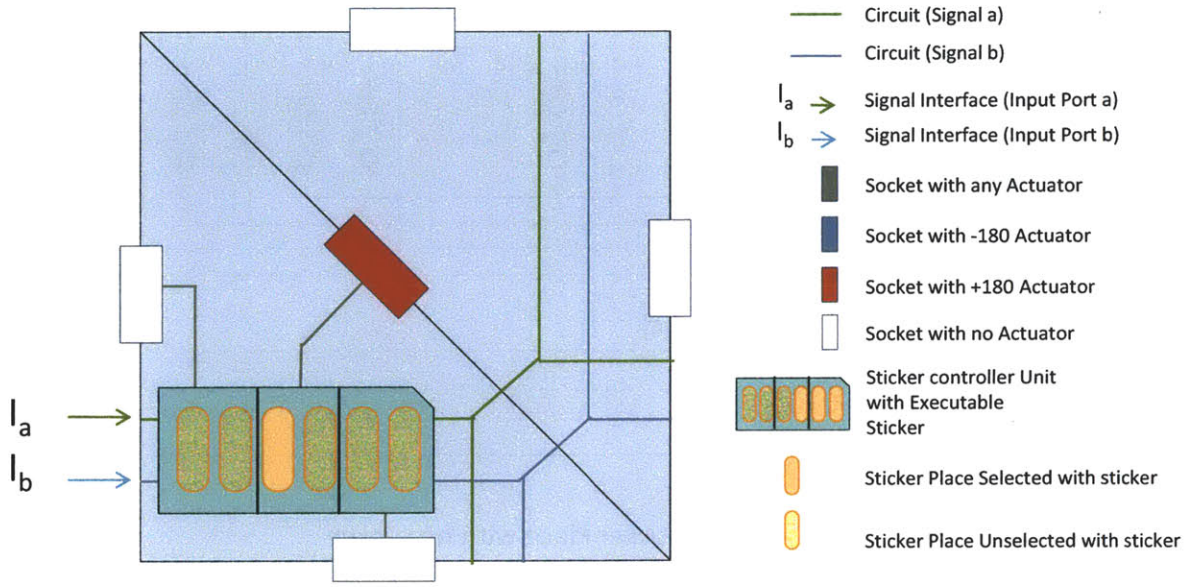


Figure 4-9: Model for the 1×1 sticker controller. The socket is connected to ground; In model diagram, the wires for ground are eclipsed (Sec. 4.1.1).

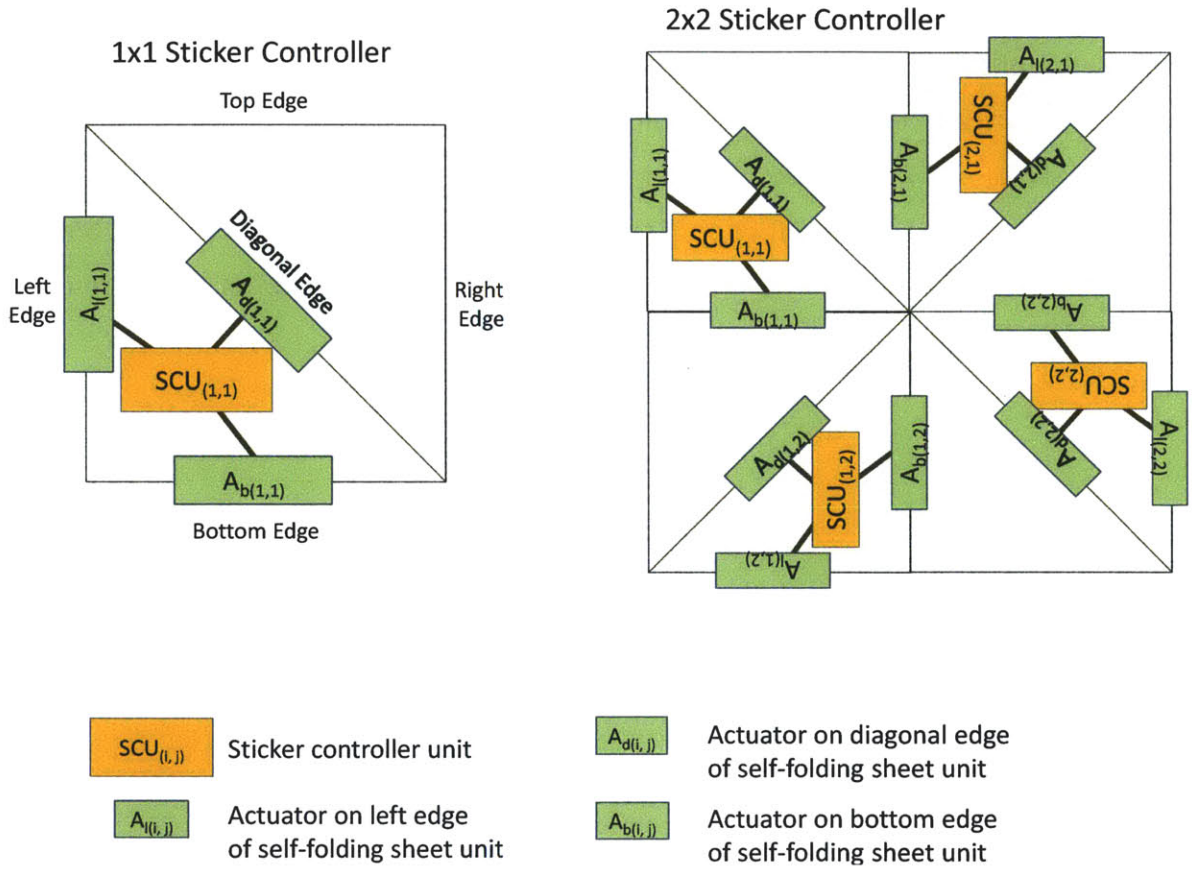


Figure 4-10: Simplified model for the 1×1 and 2×2 sticker controllers. A 2×2 self-folding sheet is composed of four 1×1 sticker controllers.

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.

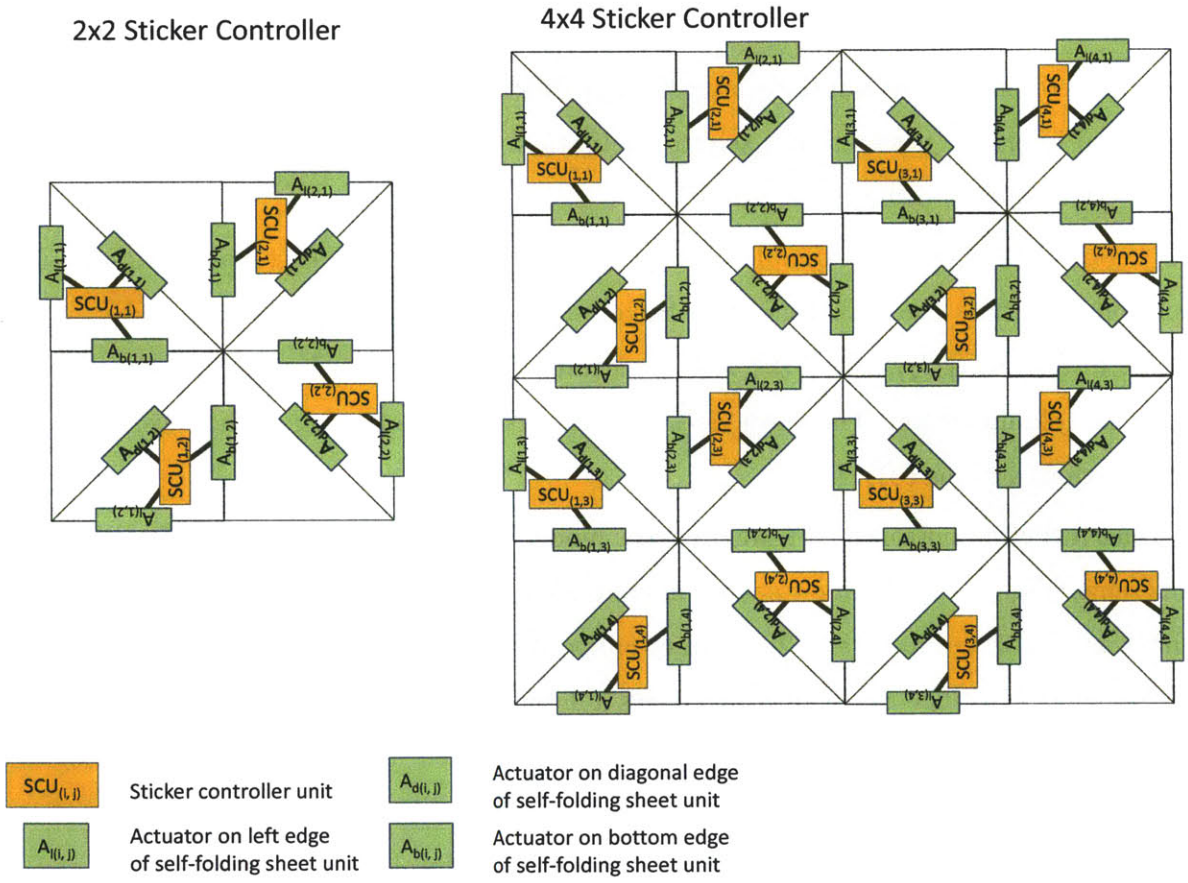


Figure 4-11: Simplified model for the 2×2 and 4×4 sticker controllers. A 4×4 self-folding sheet is composed of four 2×2 sticker controllers.

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

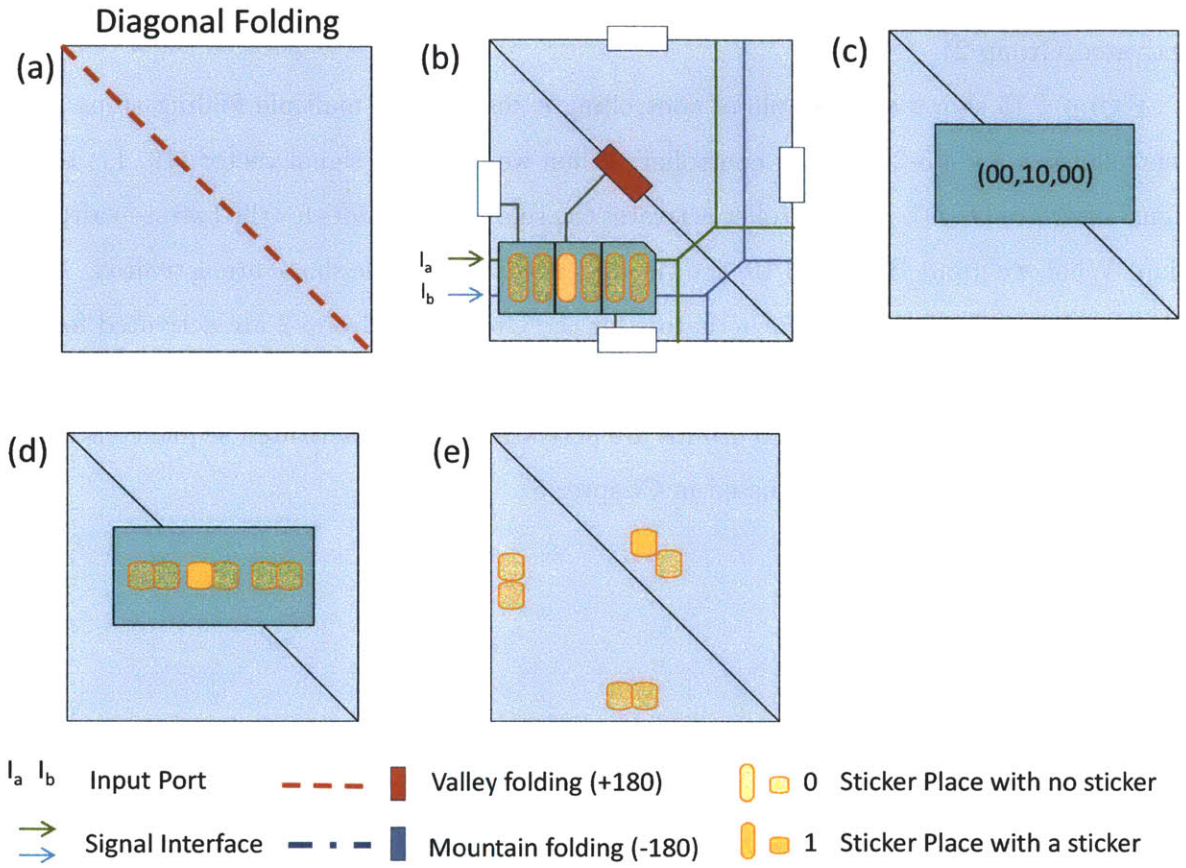


Figure 4-12: Example of 1×1 3-2-1 sticker controller with the executable sticker for a diagonal folding. (a) is the target shape diagram (diagonal folding). (b)(c)(d) and (e) show the same sticker controller with the same executable sticker. When the signal (10) is input, the controller transforms the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

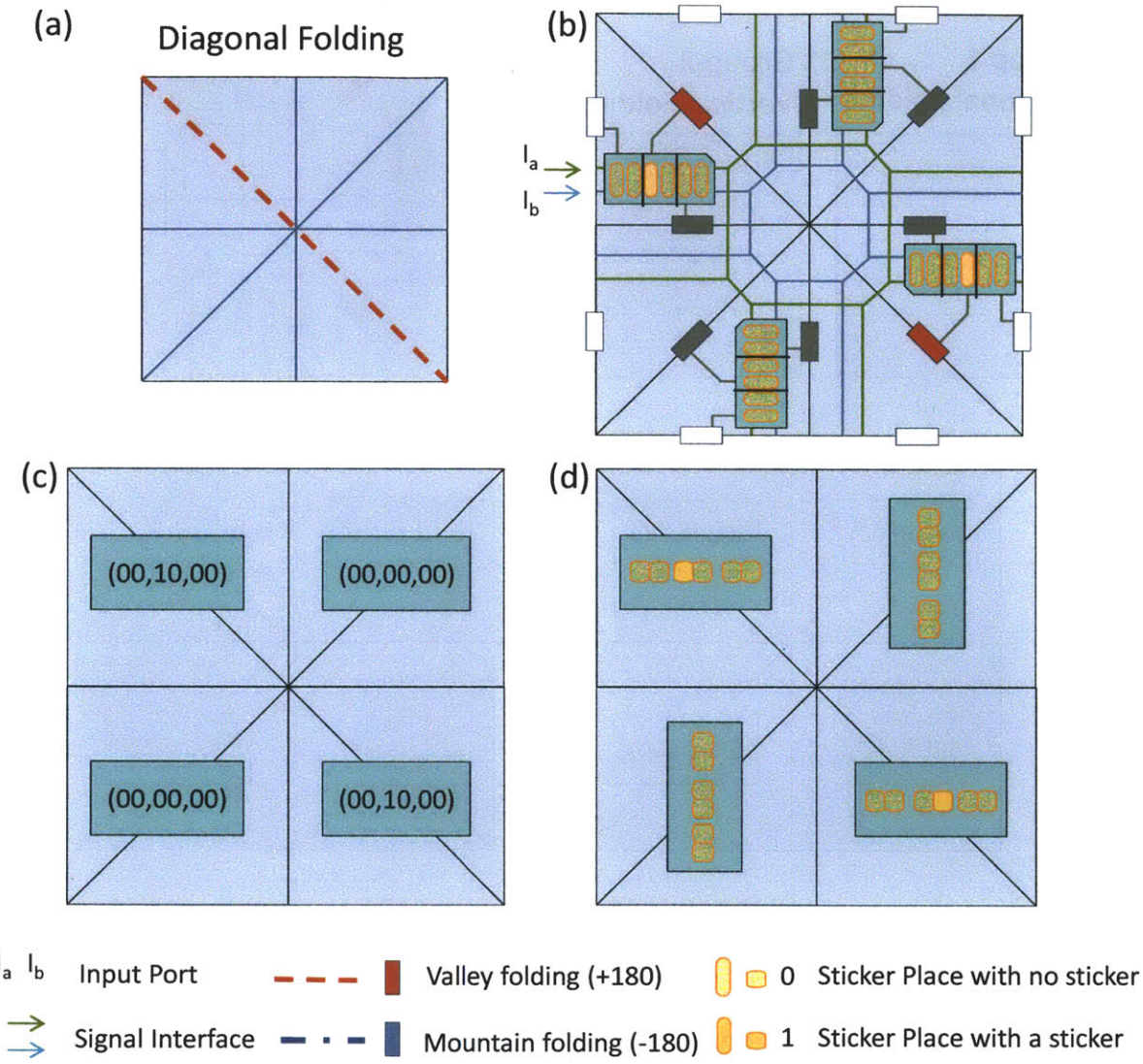


Figure 4-13: Example of 2×2 3-2-1 sticker controller with executable stickers for diagonal folding. (a) is the target shape diagram (diagonal folding). (b)(c) and (d) show the same sticker controller with the same executable sticker. When the signal (10) is input, the controller transforms the sheet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

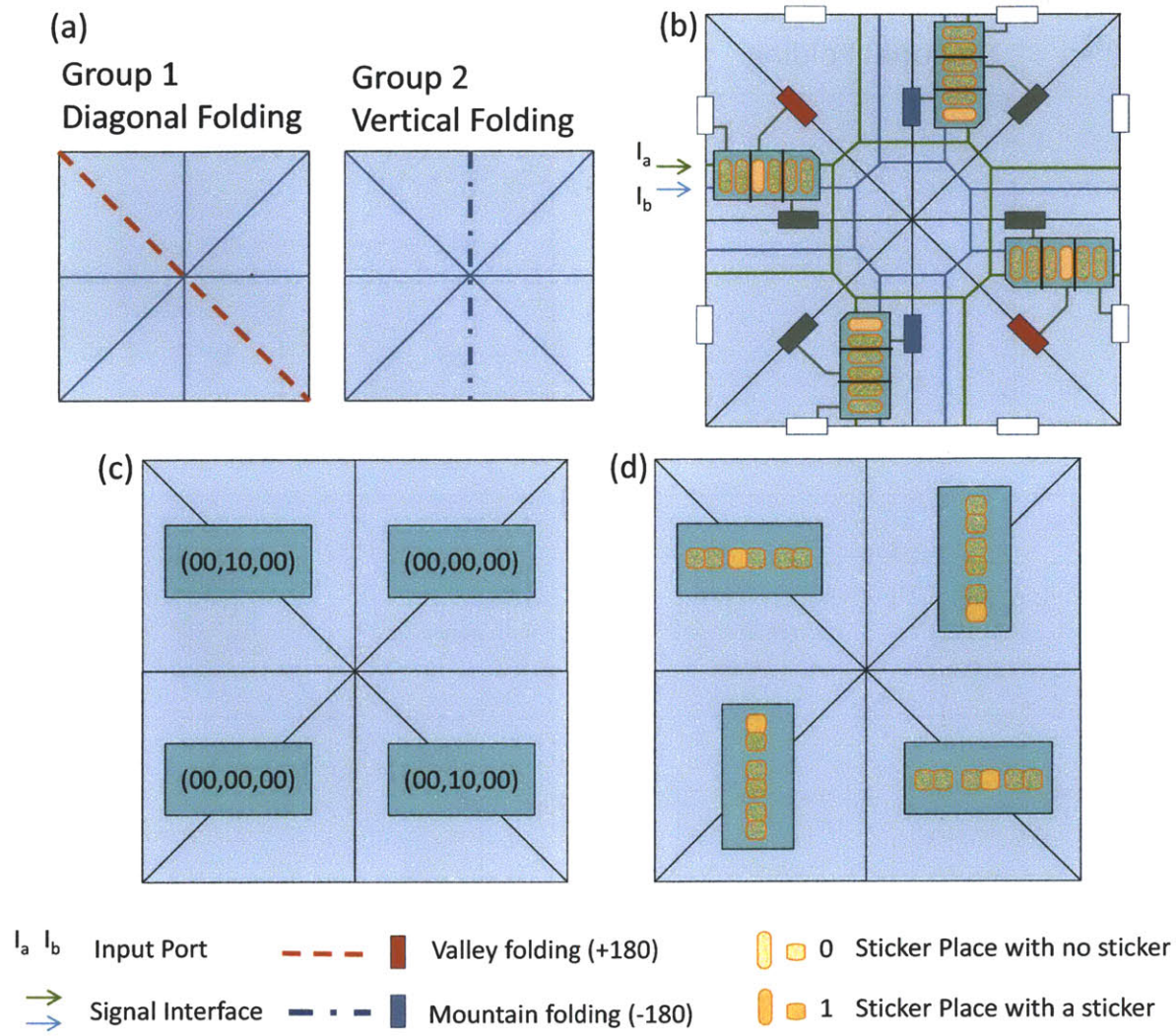


Figure 4-14: Example of 2×2 3-2-1 sticker controller with the executable sticker for multiple shape folding: (1) diagonal folding and (2) vertical folding. (b)(c) and (d) show the same sticker controller with the same executable sticker. When the signal (10) is input, the controller transforms the sheet 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.

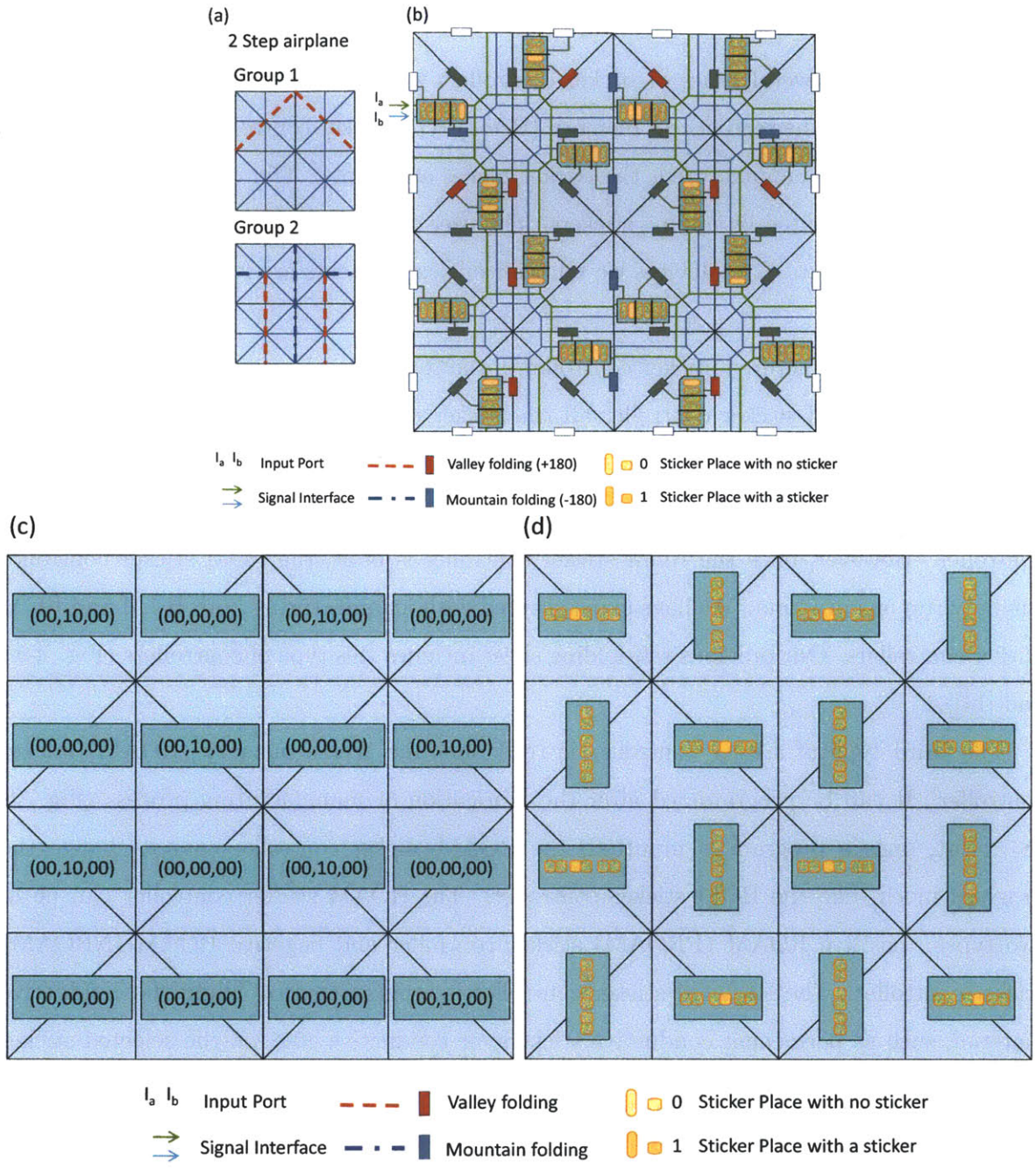


Figure 4-15: Example of 4×4 3-2-1 sticker controller with the executable sticker for the two step airplane shape. (a) is the target shape diagram. (b)(c) and (d) show the same sticker controller with the same executable sticker. When signals 10 and 11 are input sequentially, the controller transforms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

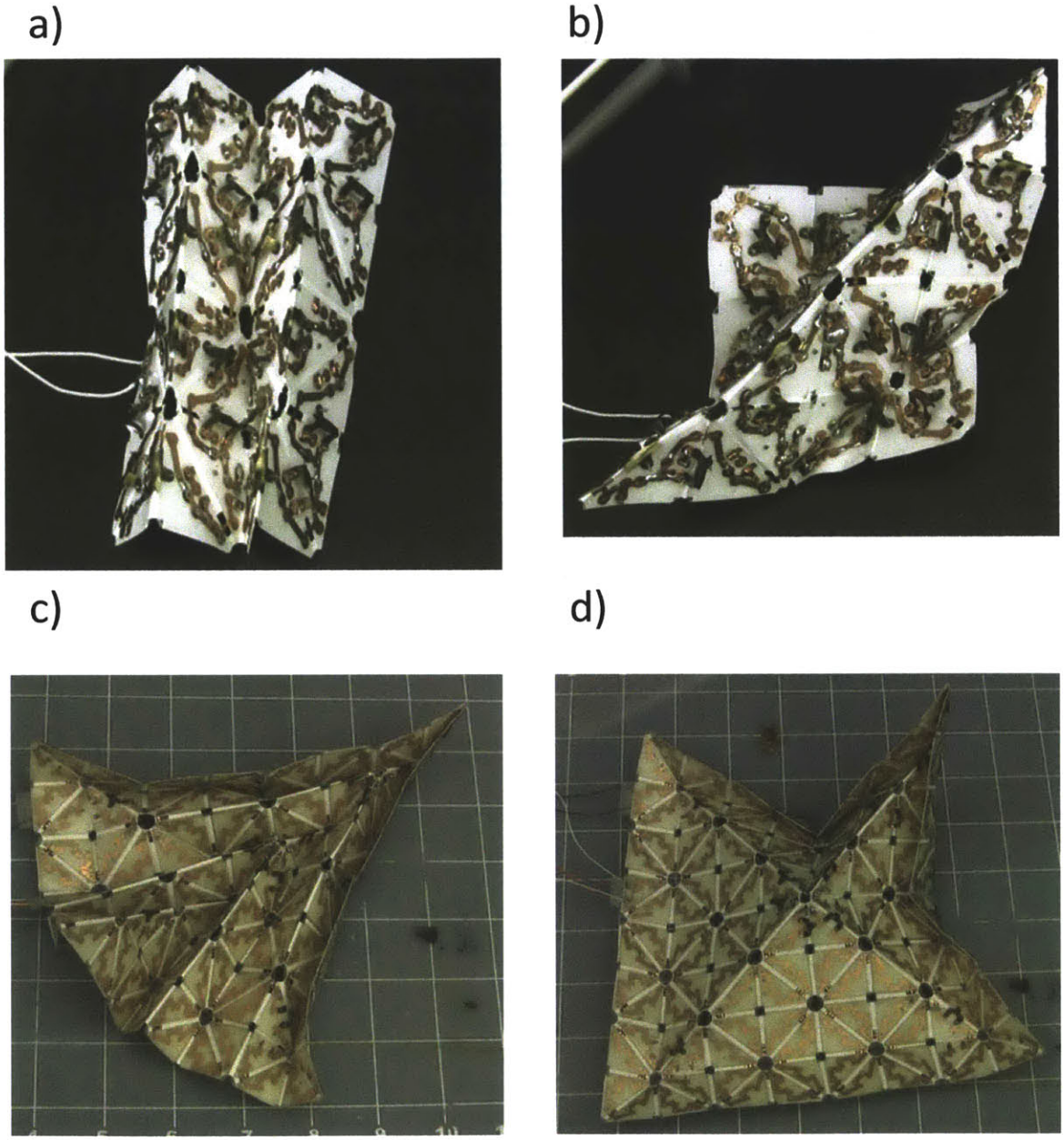


Figure 4-16: Self-Folding Sheets. (a) Vertical Folding (b) Diagonal Folding (c) Space Shuttle (d) Hat. The objects in (a) and (b) were made from a 4×4 sheet. The objects in (c) and (d) were made from an 8×8 sheet. (a)(b) 4×4 self-folding sheet. (c)(d) 8×8 self-folding sheet.

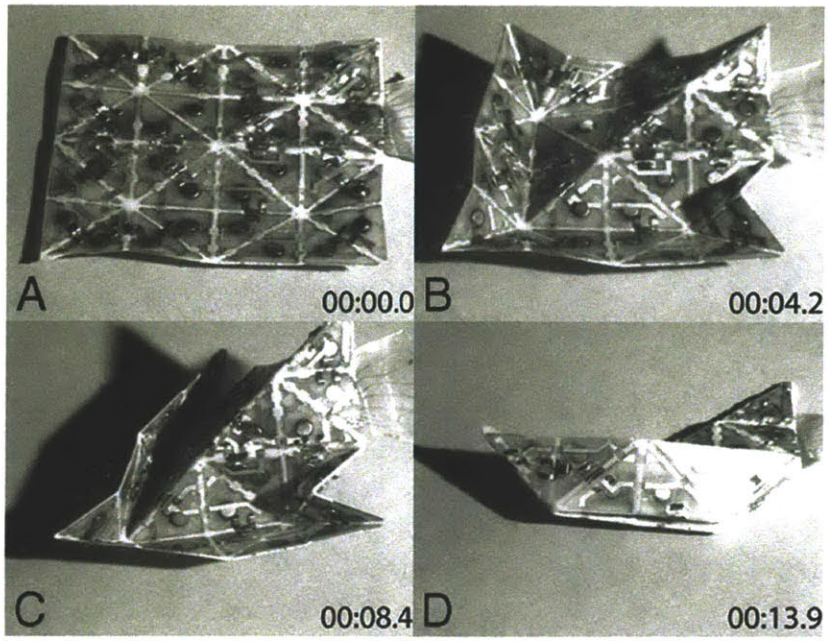


Figure 4-17: Example of read only memory (ROM) sticker controller for a boat [14]. Because the embedded controller is manually designed circuit for two shape (a boat and an airplane), we cannot change the program.

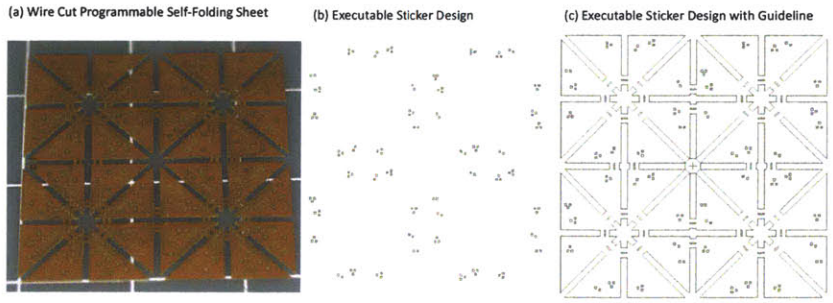


Figure 4-18: Example of negative read one-time access memory (NROAM) controller. (a) 4×4 wire cut programmable self-folding sheet (W-sheet) with no program. (b) Executable sticker design having two shapes (a space-shuttle and a pyramid). (c) Executable sticker design with guide line. The small squares 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

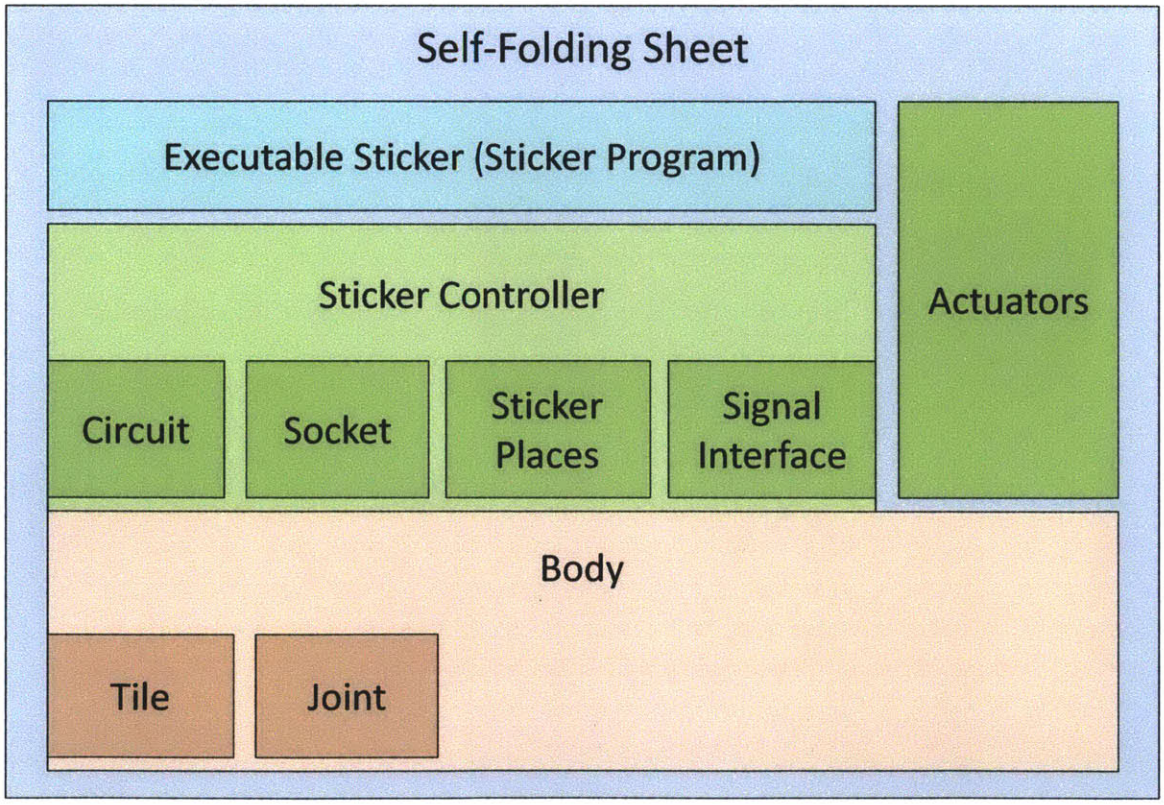


Figure 4-19: Components of Sticker Controller Architecture

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.

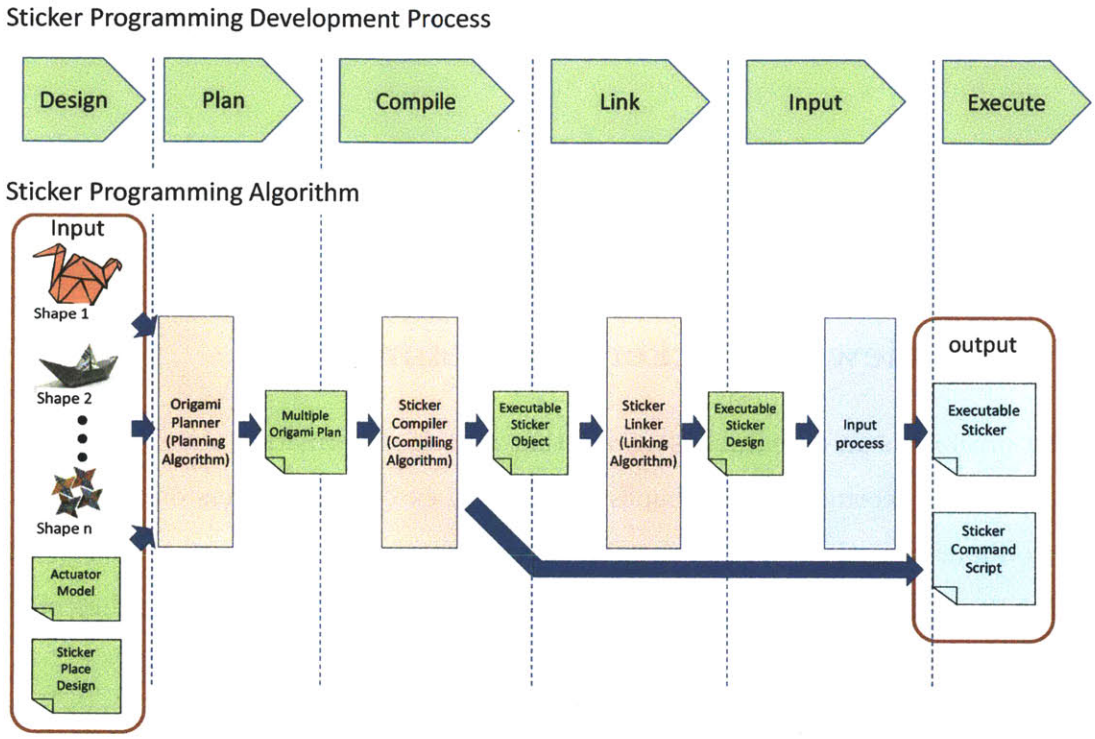


Figure 5-1: Overview of sticker programming. The diagram shows stages of sticker programming development process (top) and processes of the sticker programming algorithm for each stage (bottom). (Design Stage) The user selects the design. (Plan Stage) The planner computes the folding sequence and the 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.

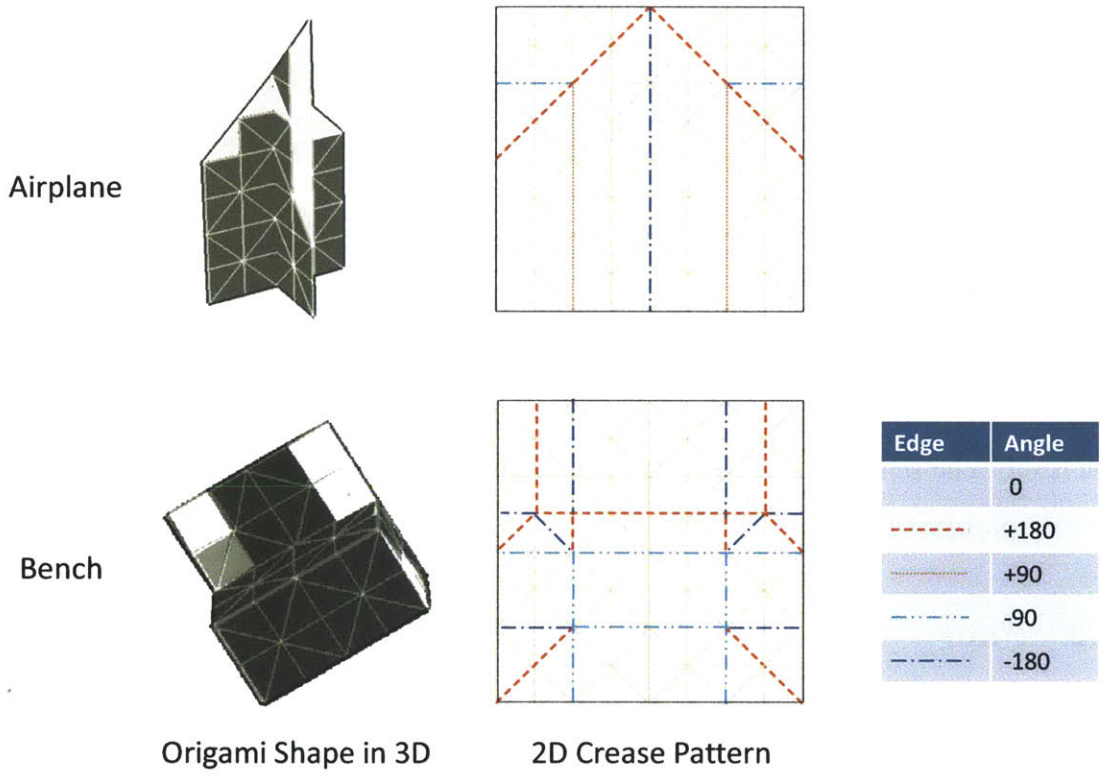


Figure 5-2: Target shapes of airplane and bench drawn in 2D crease pattern and the crease patters (with mountain and valley folds) for actuating the shapes. A multi-origami sheet will include both of these crease patterns.

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* $1 \leq i \leq n$. $s_j = \{ g \mid g \in G$ *for* j^{th} *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 $\cup$ 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 $\cup$ 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 $\cup$ Group 3 is the same as the time step t_2 of Shape 1, top-right; Group 3 $\cup$ Group 4 is the same as Shape 2, bottom-right.

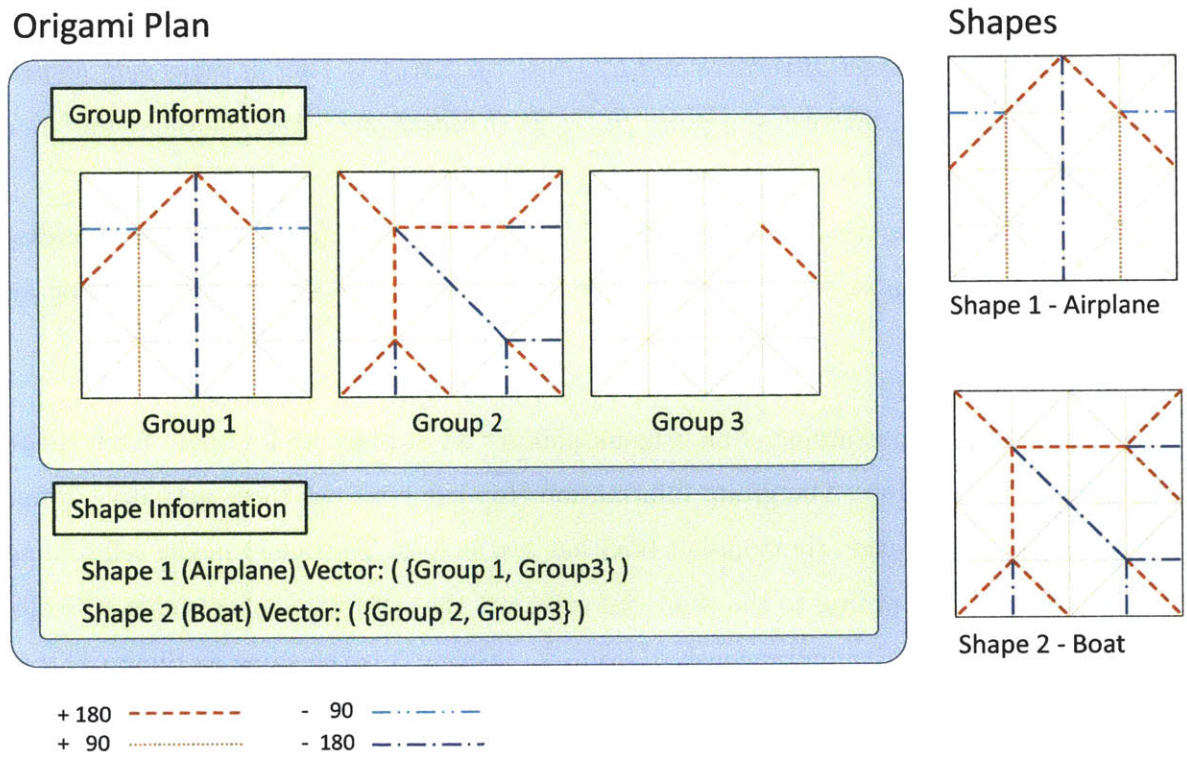


Figure 5-3: Origami plan for airplane and boat. All edges in group i are folded in parallel. Different groups are controlled in sequence.

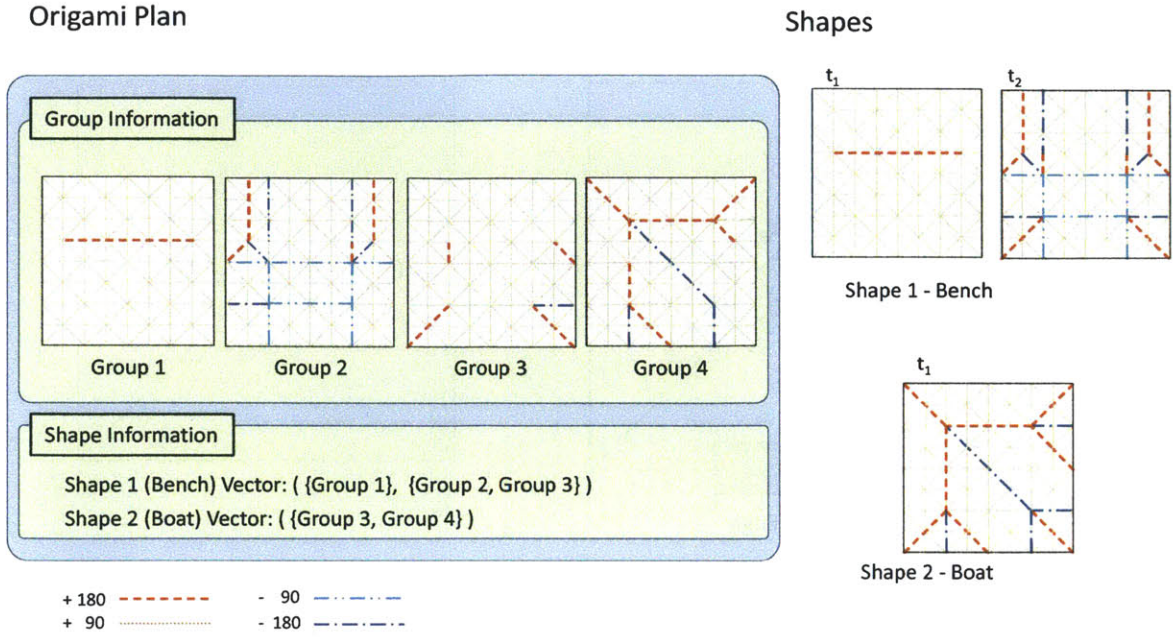


Figure 5-4: Origami plan for bench and boat.

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, $\delta : \Sigma \rightarrow A$.

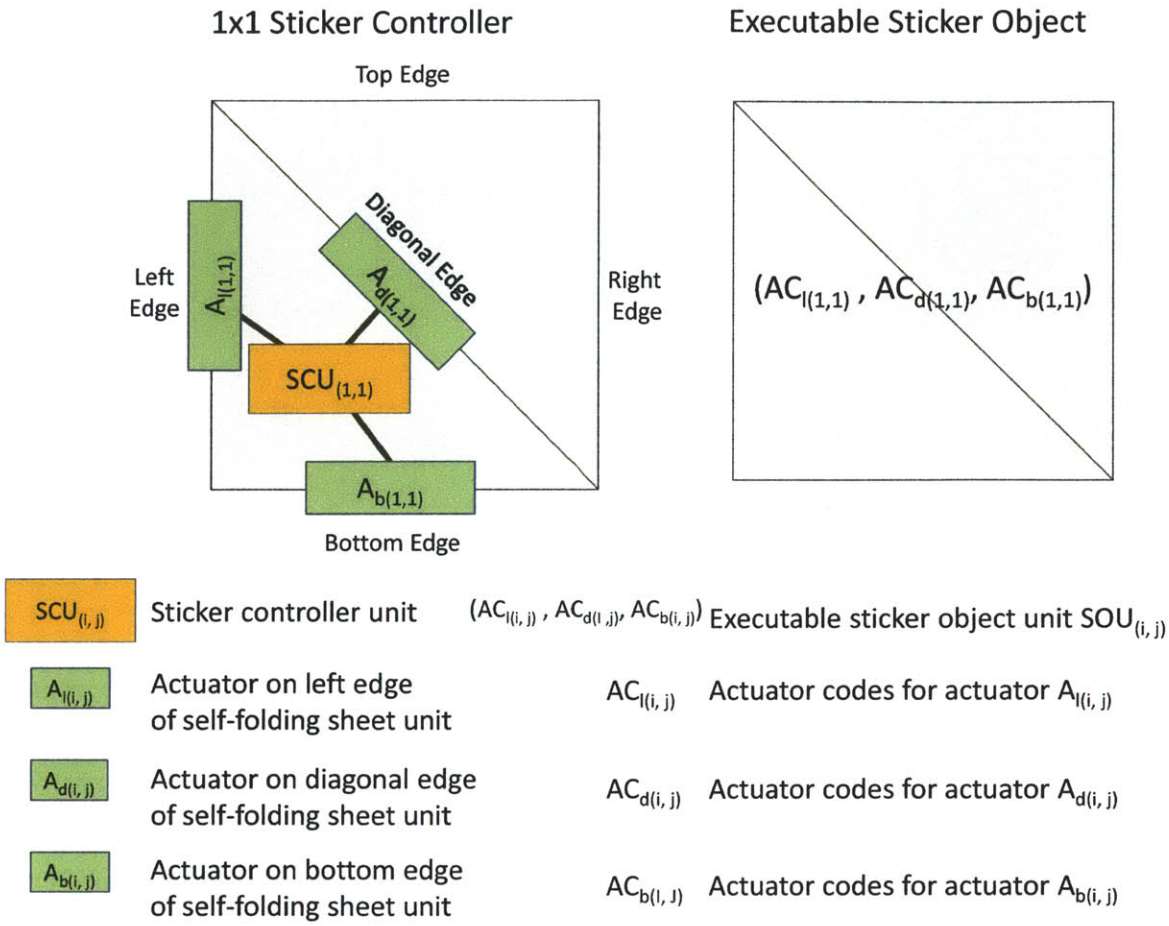


Figure 5-5: Executable sticker object for 1×1 sticker controller. (left) Simplified model of 1×1 sticker controller. A sticker controller unit controls three actuators. (right) Corresponding executable sticker object. Each element of the 3-tuple of the object has actuator codes for each actuator.

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).

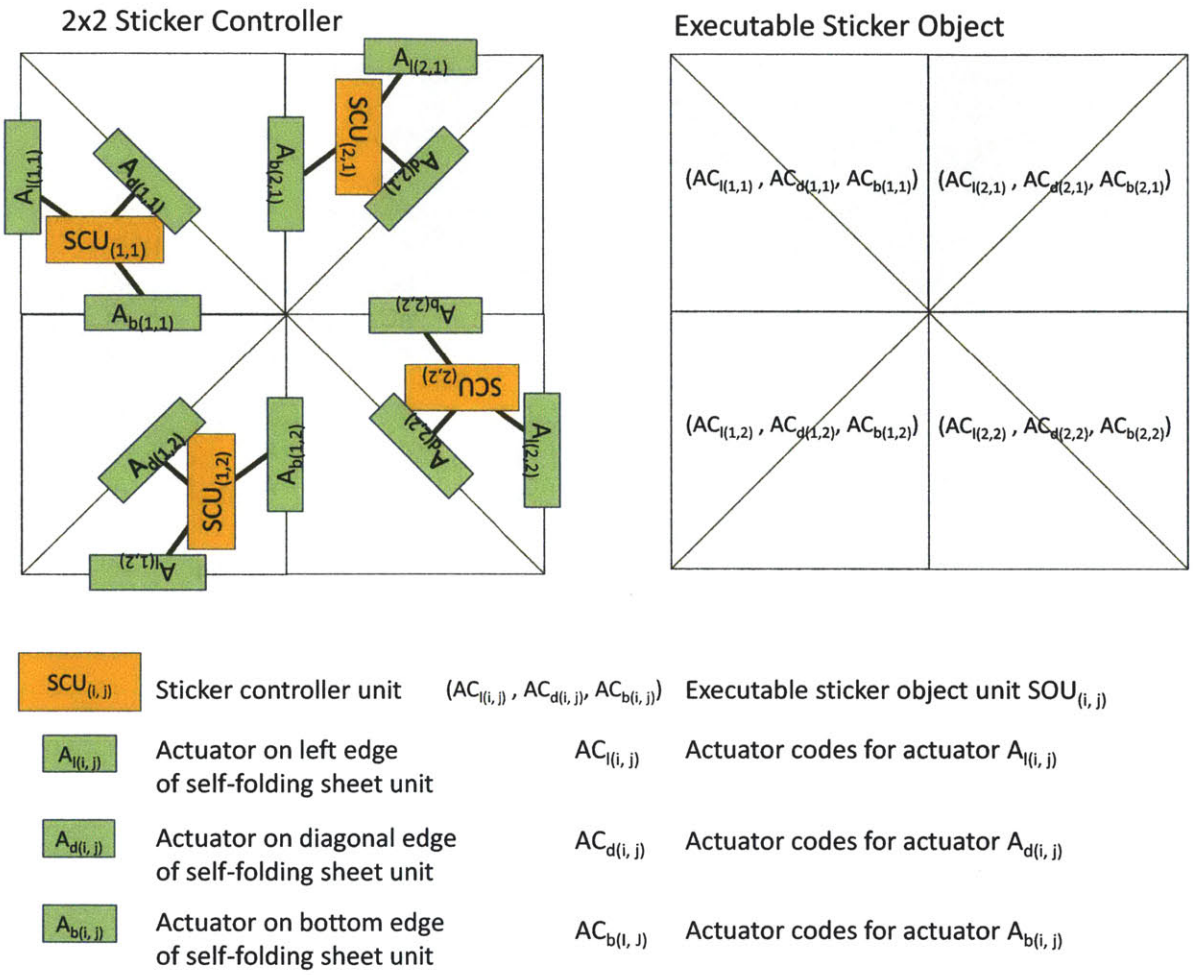


Figure 5-6: Executable sticker object for 2×2 sticker controller. (left) Simplified model of 2×2 sticker controller composed of four 1×1 sticker controllers. Each sticker controller unit on the 1×1 sticker controller controls three actuators. (right) Corresponding executable sticker object. Each element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

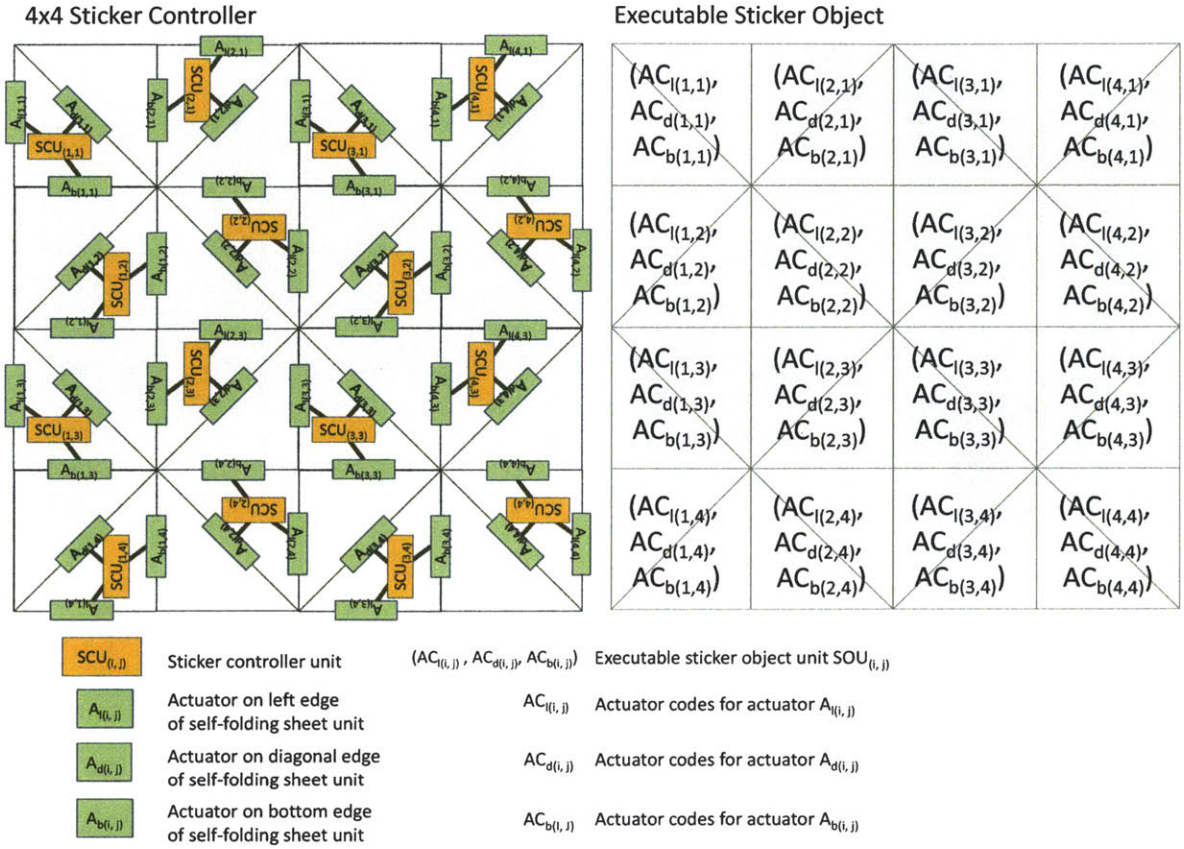


Figure 5-7: Executable sticker object for 4×4 sticker controller. (left) Simplified model of 4×4 sticker controller composed of 16 1×1 sticker controllers. Each sticker controller unit on the 1×1 sticker controller controls three actuators. (right) Corresponding executable sticker object. Each element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

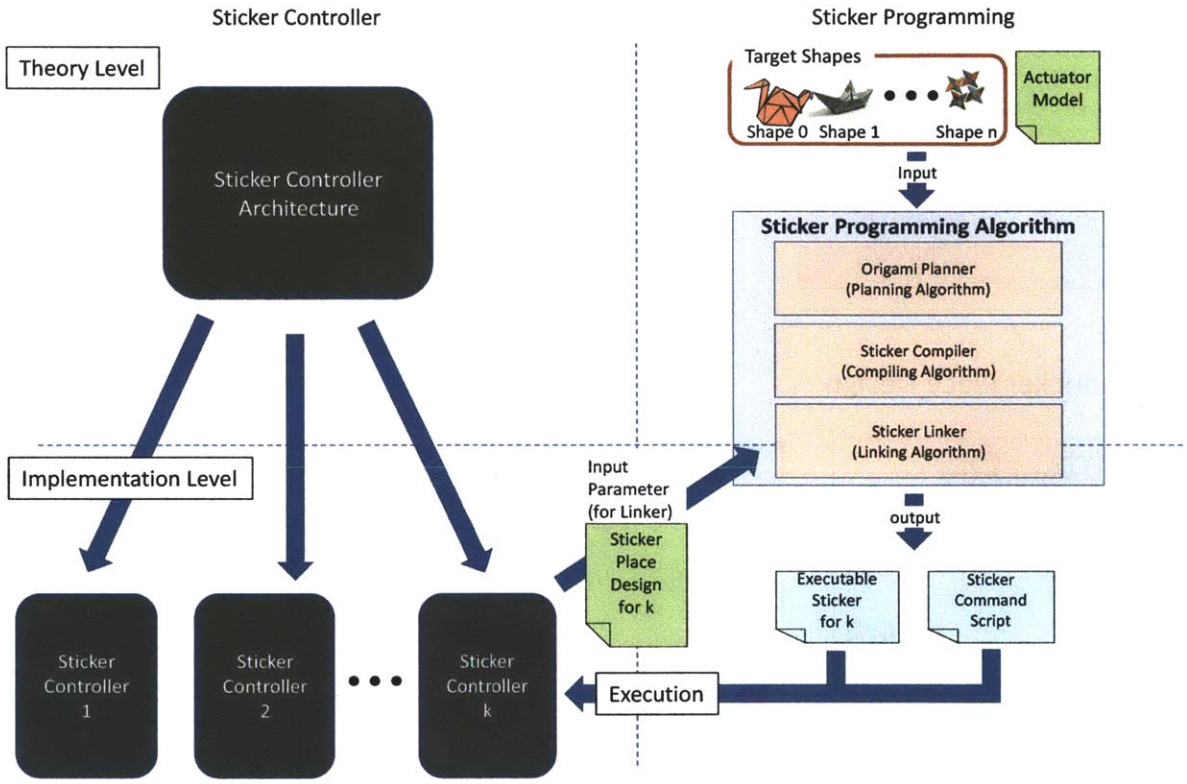


Figure 5-8: Overview of the approach for programming and controlling the self-folding sheet. The sticker place design is used as a parameter of a type of self-folding sheet.

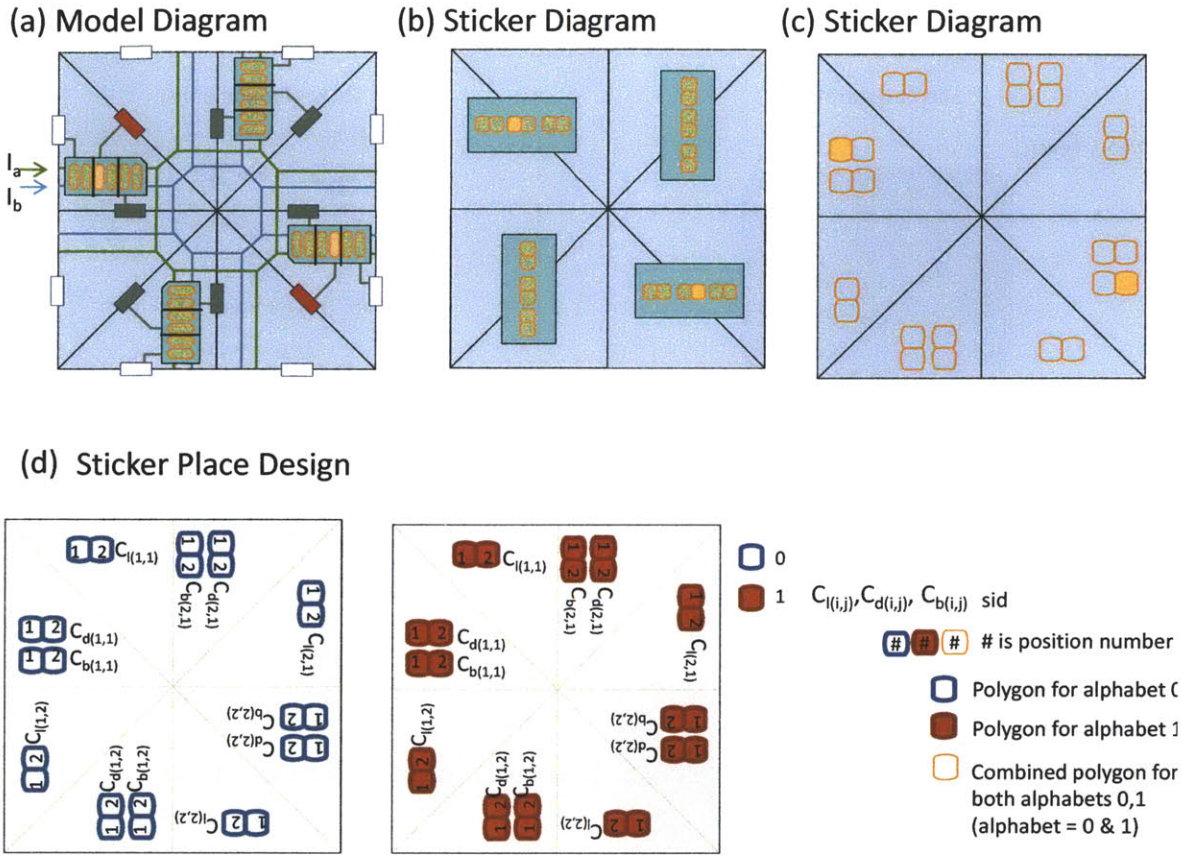


Figure 5-9: Example of a sticker place design for a 2×2 sticker controller. (a)(b)(c) present a same model for the 2×2 sticker controller in three different diagrams. (d) presents the sticker place design for the model. The diagrams (a) and (b) do not contain the locations of the sticker places. The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

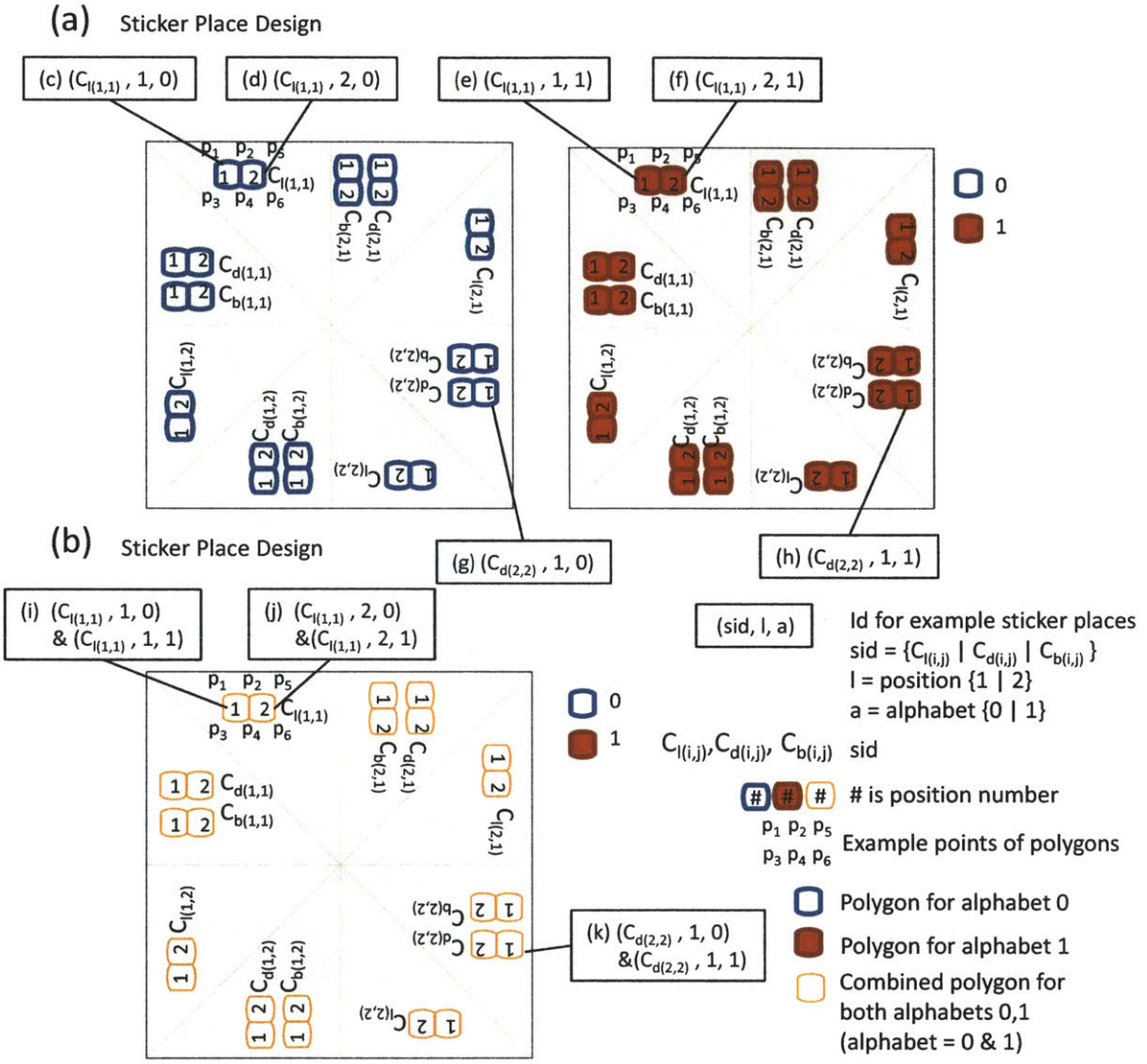
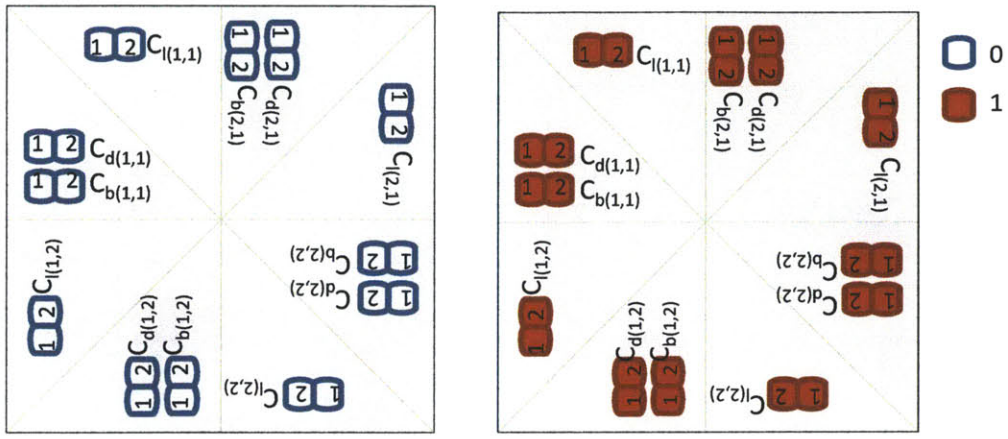


Figure 5-10: Example of sticker place design for 2×2 self-folding sheet.
(a) and (b) present a same sticker place design. (a) is a detailed diagram for the sticker place design. (b) is a simplified diagram for the sticker place design. The boxes of (c)(d)...(k) have the ids of the sticker places.

(a) Sticker Place Design



(b) Sticker Place Design

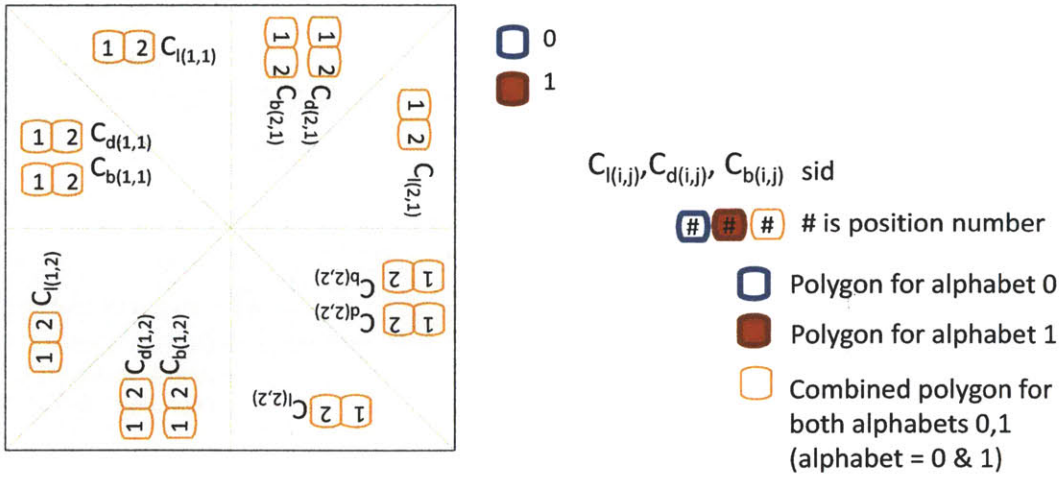


Figure 5-11: Example of sticker place design with no label for 2×2 self-folding sheet. (a) and (b) present a same sticker place design. (a) is a detailed diagram for the sticker place design. (b) is a simplified diagram for the sticker place design.

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).

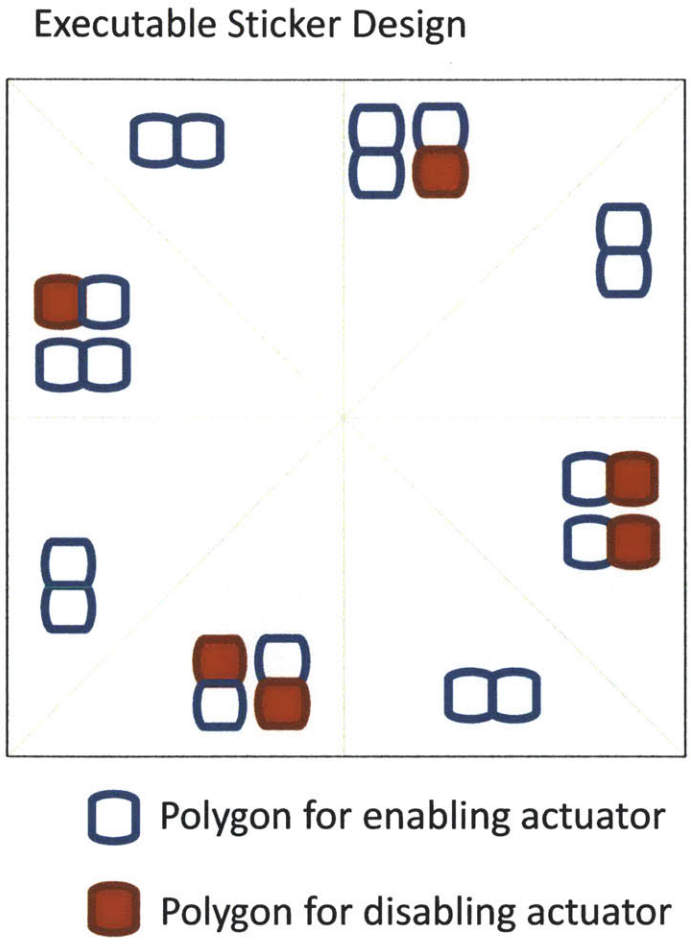


Figure 5-12: Example of executable sticker design. The executable sticker design is the result of a sticker linking example discussed in Section 5.3.3. The red square is a sticker place enabling its actuator. The blue square is a sticker place disabling its actuator.

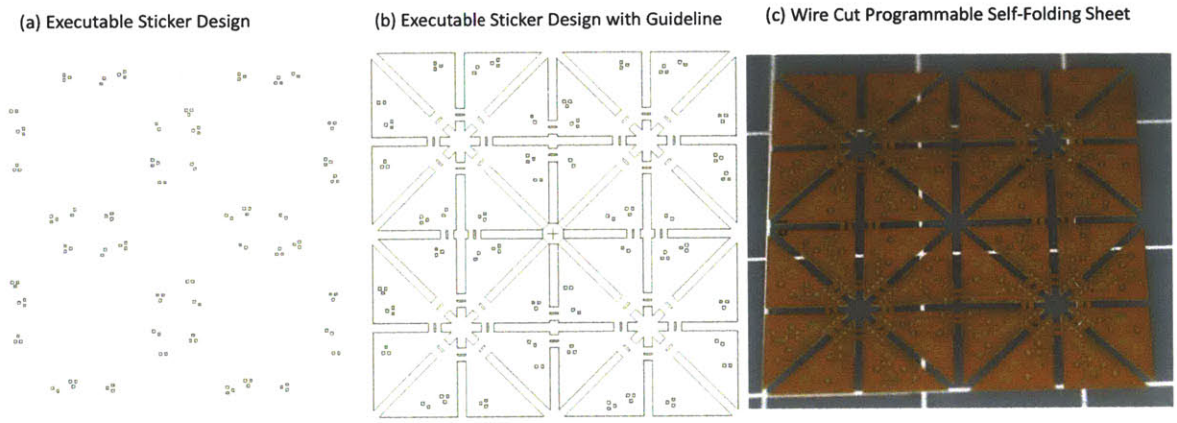


Figure 5-13: Example of an executable sticker design for an implemented self-folding sheet. (a) Executable sticker design for two shapes (a space-shuttle and a pyramid). By manually computing the sticker linker, we generate the executable sticker design. (b) Executable sticker design with guide line. 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.

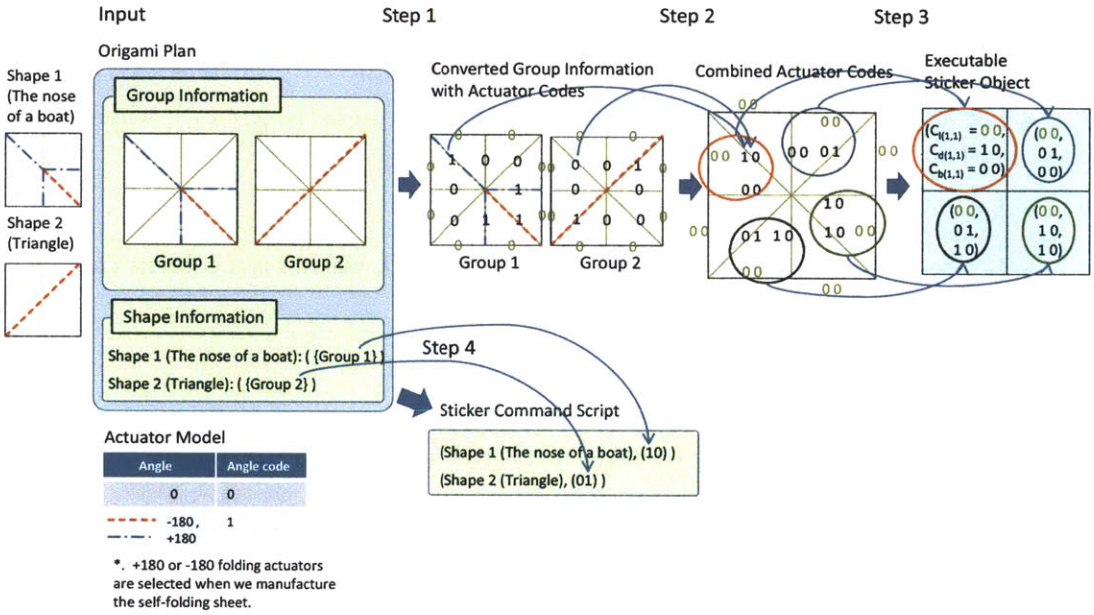


Figure 5-15: Compiling a sticker for 2×2 sticker controller for the nose of a boat and a triangle (Overview of Fig. 5-14). Given origami plan (input), the sticker compiler constructs an executable sticker object and a sticker command script. Each step of the example is shown in much detail in Figures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

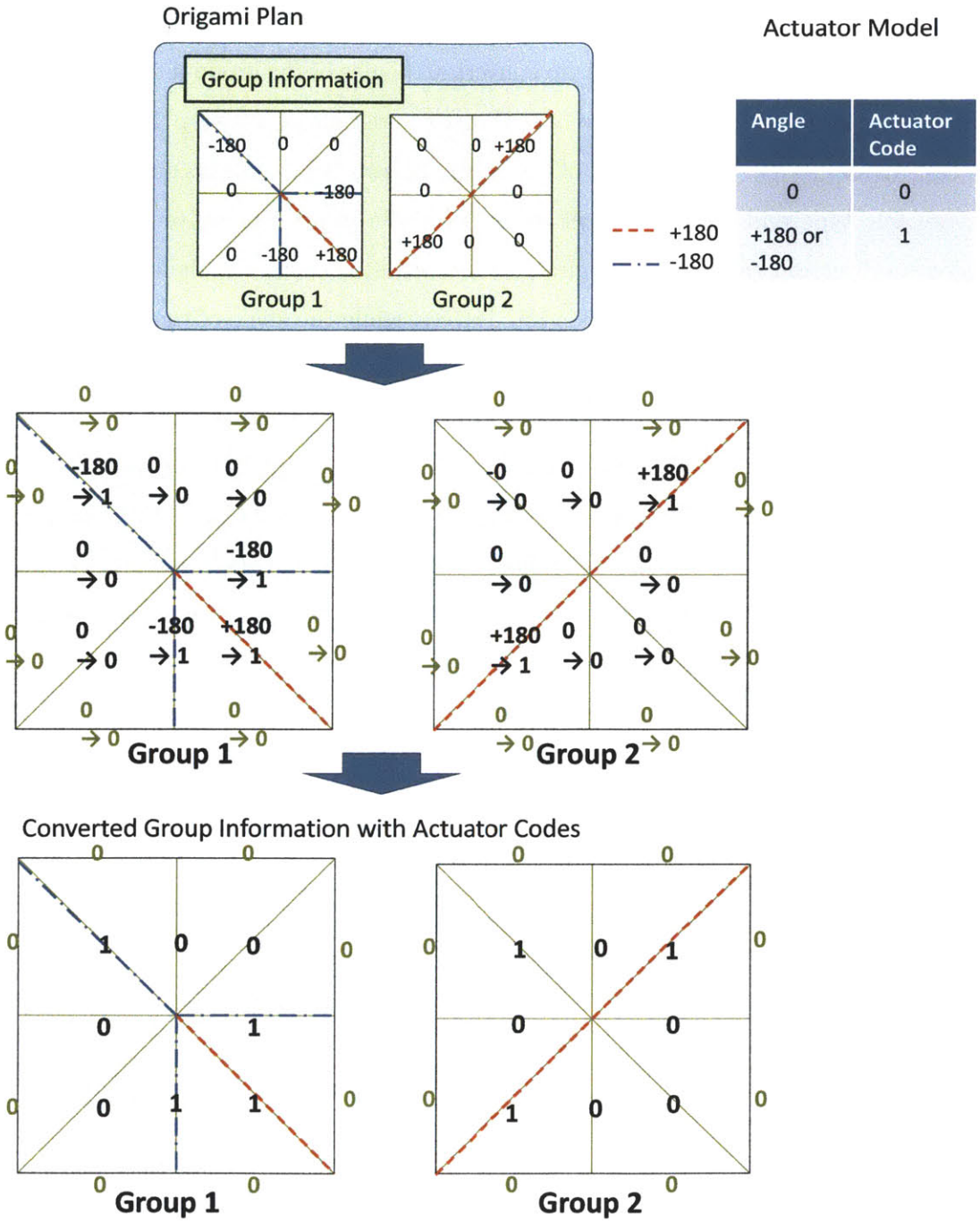


Figure 5-17: Compiling a sticker object for 2×2 sticker controller the nose of a boat and a triangle (Step 1 on Fig. 5-14). Given the group information (Group1 and Group2) of the origami plan and the actuator model (top), the algorithm converts the angle of each edge to the actuator code (middle). The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

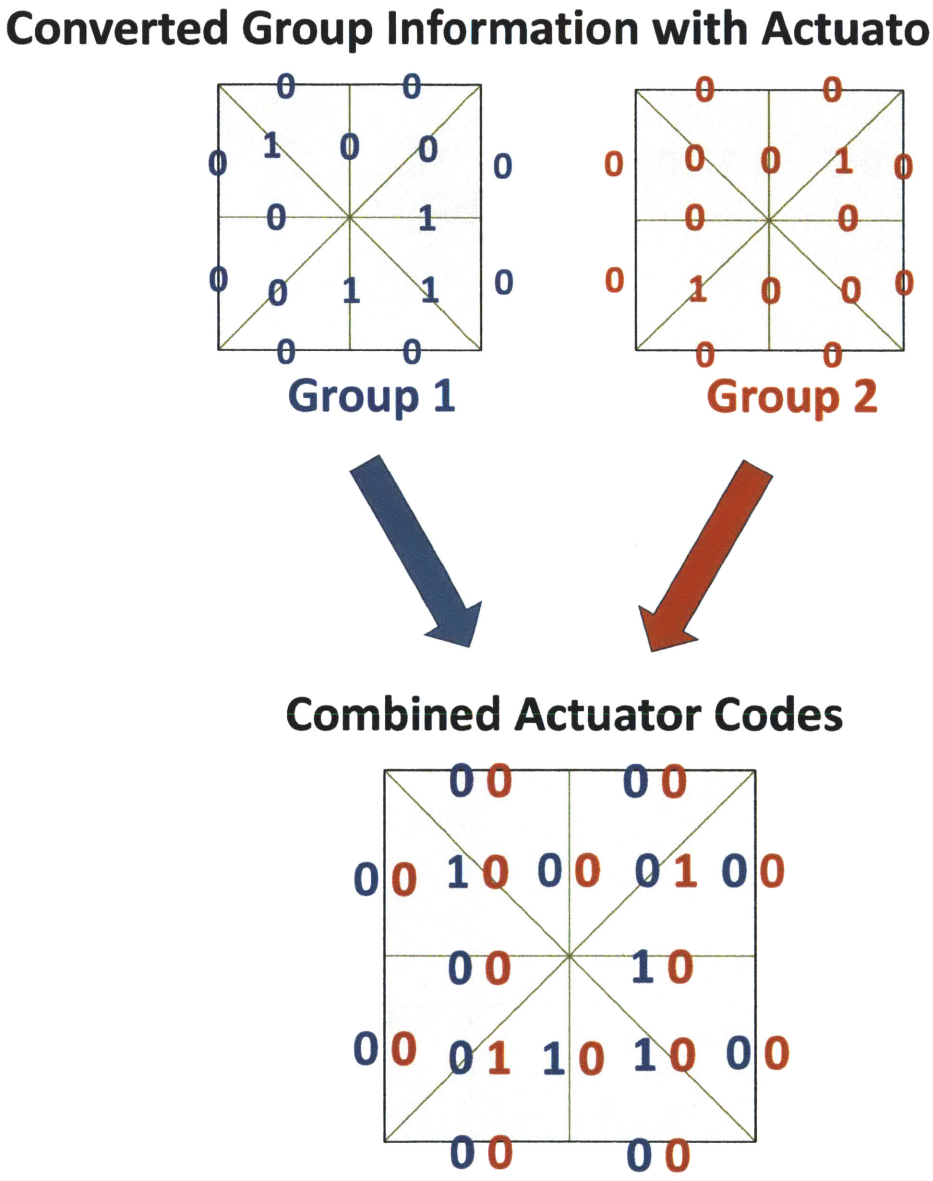


Figure 5-18: Compiling a sticker object for 2×2 sticker controller for the nose of a boat and a triangle (Step 2 on Fig. 5-14). Given the converted group information with actuator codes (top) from step 1 (Fig. 5-17), the algorithm combines the actuator codes (bottom). The result of step 2 is the combined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

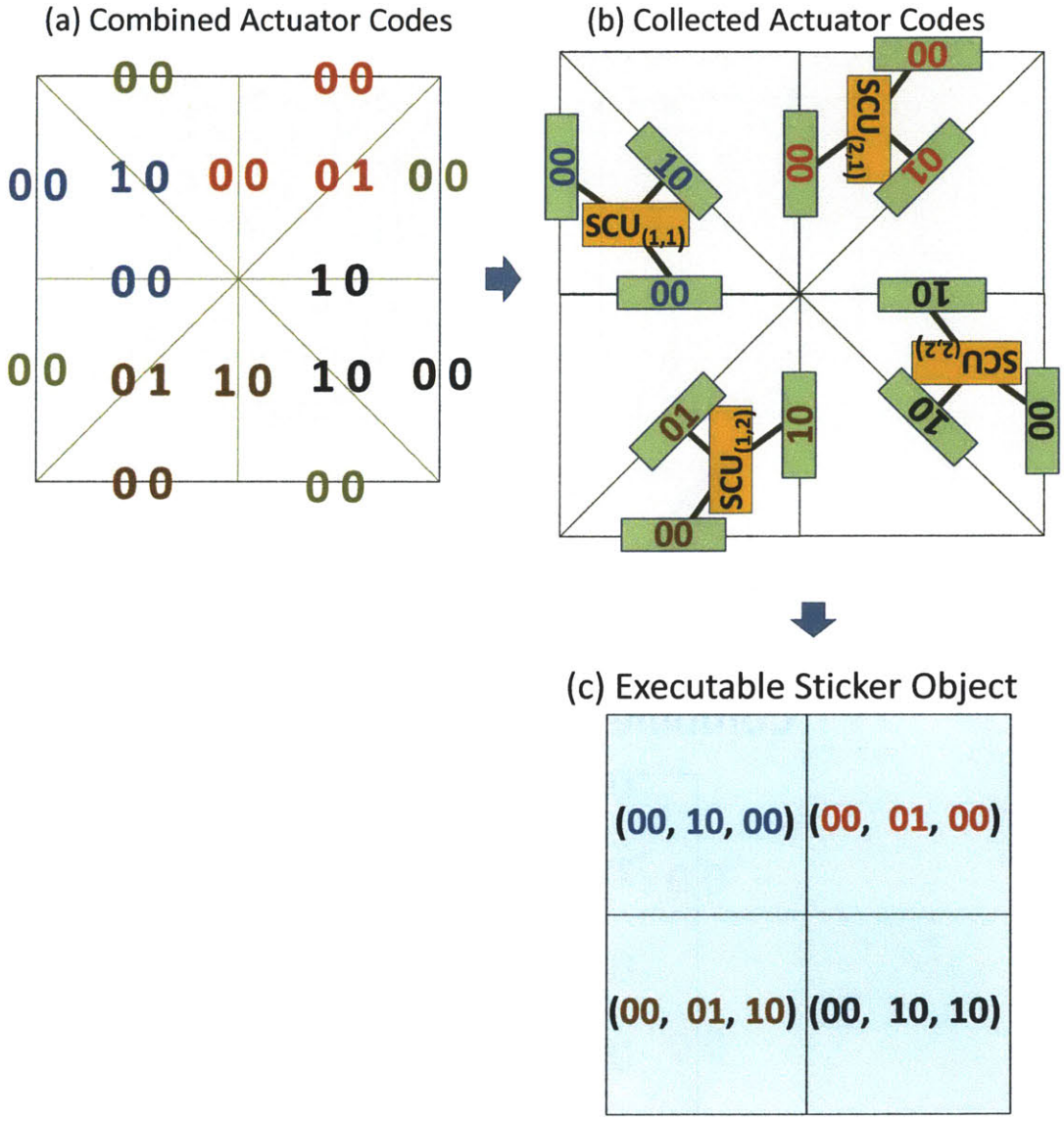


Figure 5-19: Compiling a sticker object for 2×2 sticker controller for the nose of a boat and a triangle (Step 3 on Fig. 5-14, Fig. 5-16). Given the combined actuator codes (a) from step 2 (Fig. 5-18), the algorithm builds the executable sticker object (c). (b) shows corresponding actuator codes for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

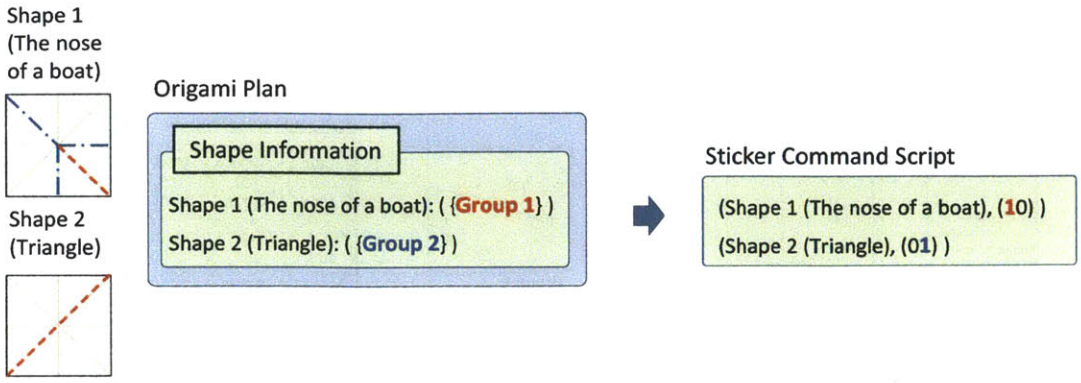


Figure 5-20: Compiling a sticker object for 2×2 sticker controller for the nose of a boat and a triangle (Step 4 on Fig. 5-14). Shape 1 of the shape information of the given origami plan (Fig. 5-15) is converted to the first line of the sticker command script. Shape 2 of the shape information of the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

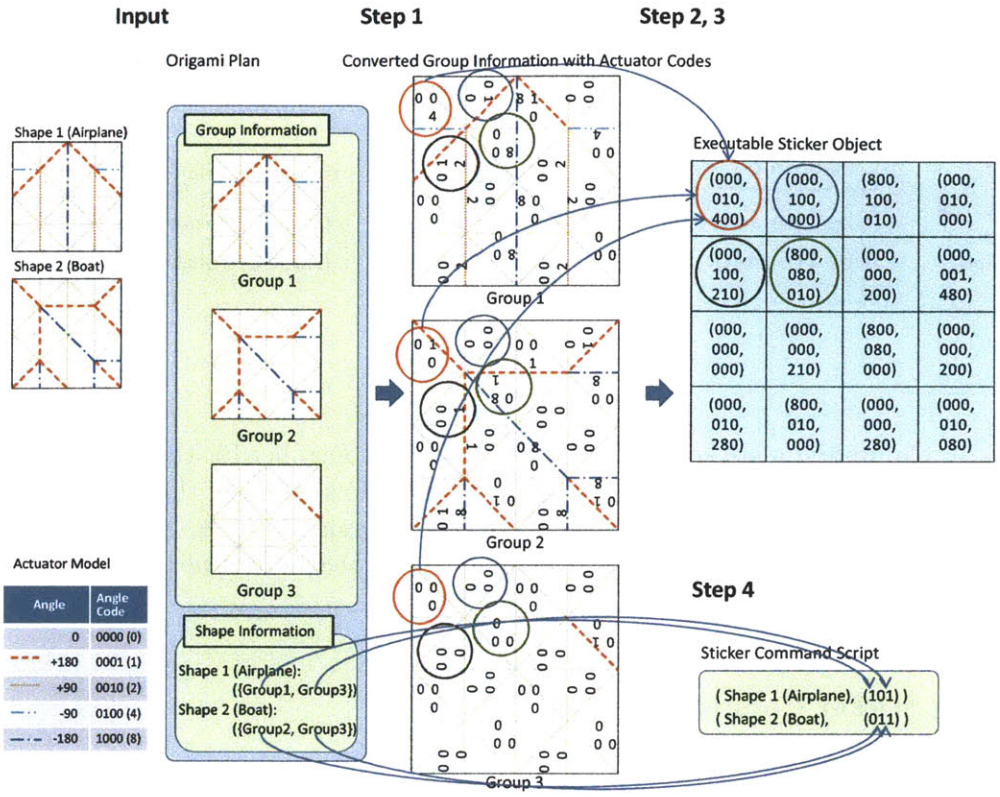


Figure 5-21: Compiling a sticker object for 4×4 sticker controller for an airplane and a boat (Overview of Fig. 5-14).

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.

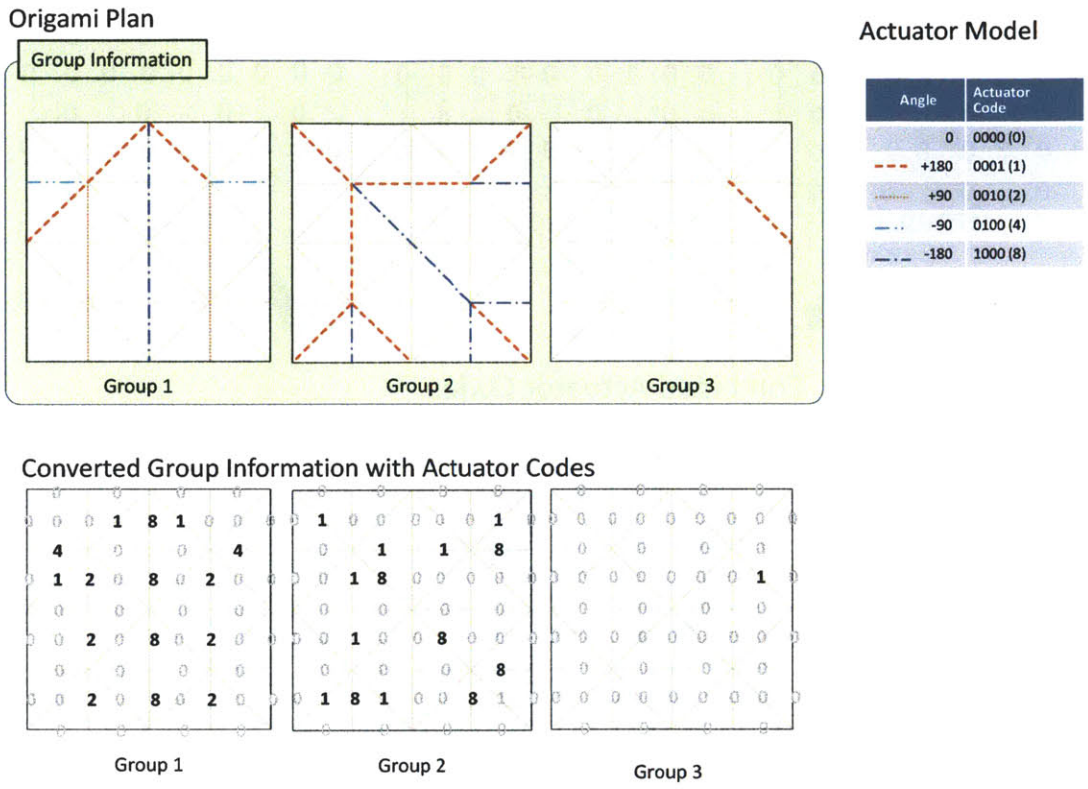


Figure 5-22: Compiling a sticker object for 4×4 sticker controller for an airplane and a boat (Step 1 on Fig. 5-14). Given the origami plan and the actuator model, the algorithm generates a converted group information with actuator codes. The bold numbers are converted numbers from the angles that is 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 ( ).

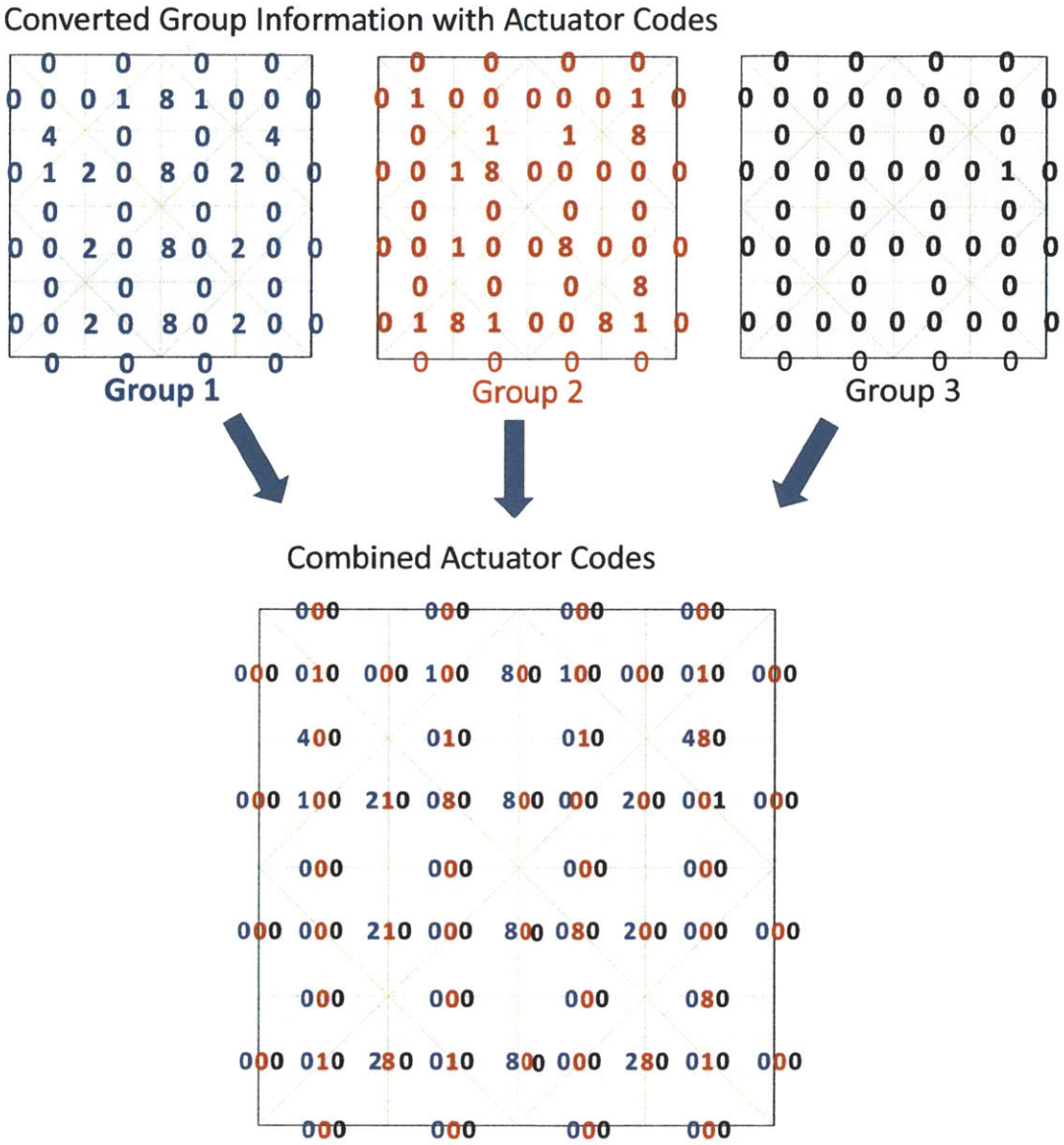
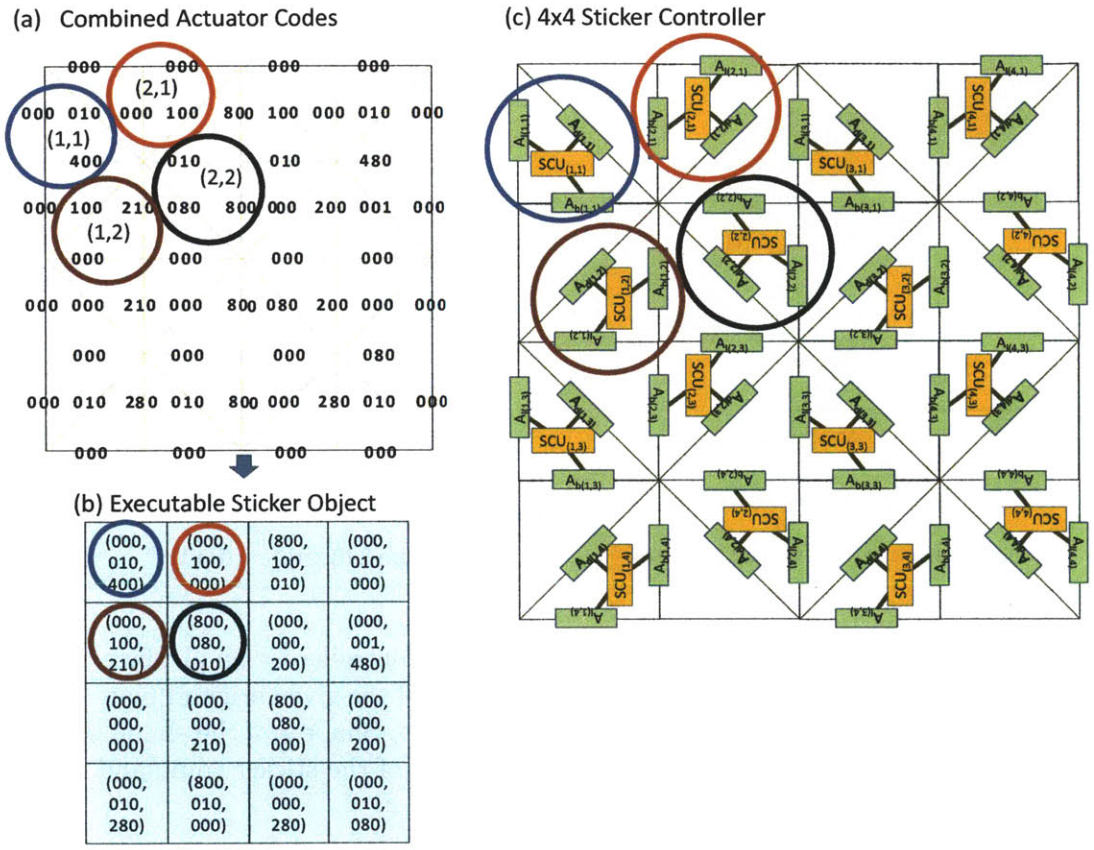


Figure 5-23: Compiling a sticker object for 4×4 sticker controller for an airplane and a boat (Step 2 on Fig. 5-14). Given the converted group information with actuator codes (Fig. 5-22), the algorithm combines actuator codes of each group. The red numbers of combined actuator codes are from Group 1.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

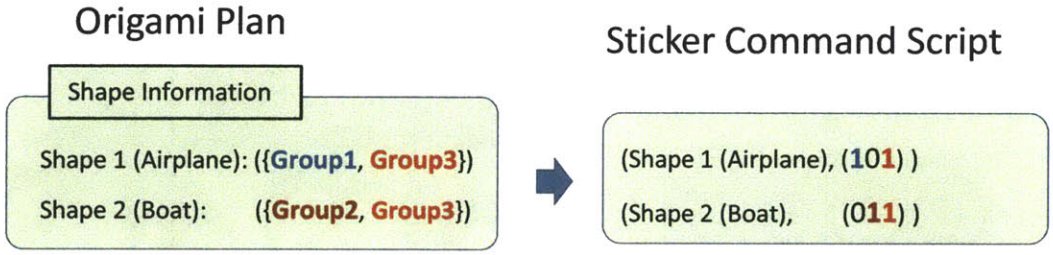


Figure 5-25: Compiling a sticker object for 4×4 sticker controller for an airplane and a boat (Step 4 on Fig. 5-14). Shape 1 of the shape information of the given origami plan (Fig. 5-21) is converted to the first line of the sticker command script. Shape 2 of the shape information of the given origami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

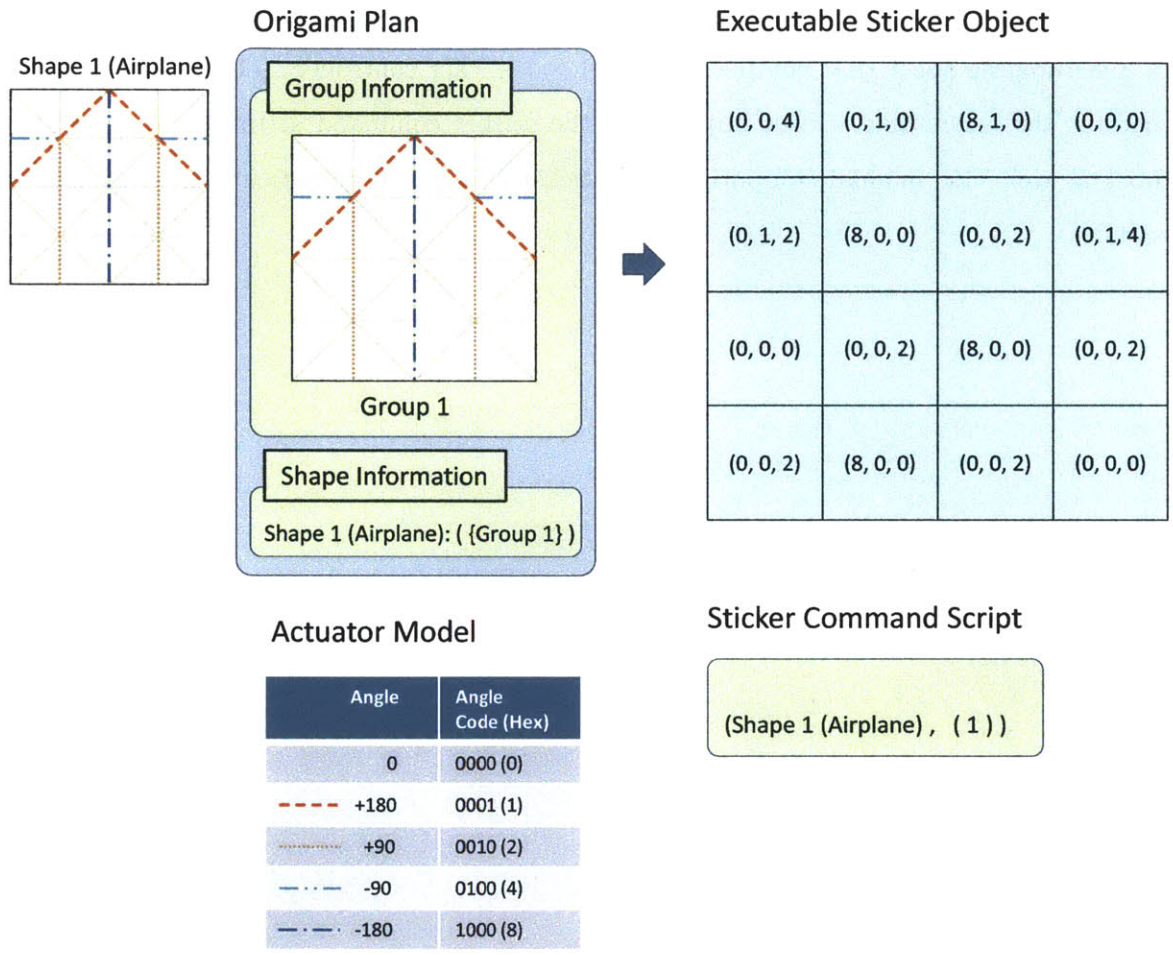


Figure 5-26: Result of the sticker compiler for a single shape, an airplane. The actuator codes of the executable sticker object is hex. The hex actuator codes of the actuator model are in ( ).

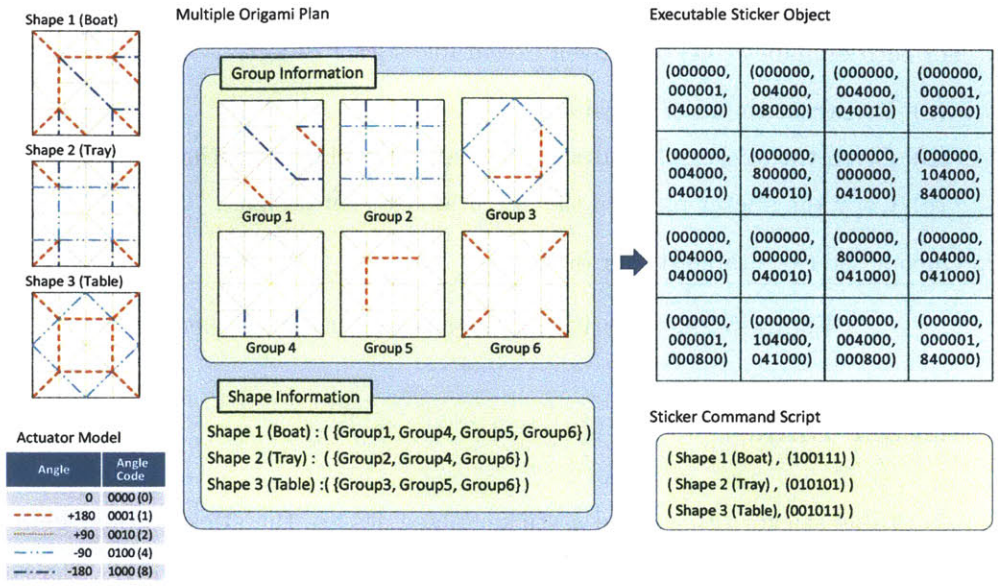


Figure 5-27: Result of the sticker compiler for three shapes, a boat, a tray and a table. The actuator codes of the executable sticker object is hex. The hex actuator codes of the actuator model are in ( ).

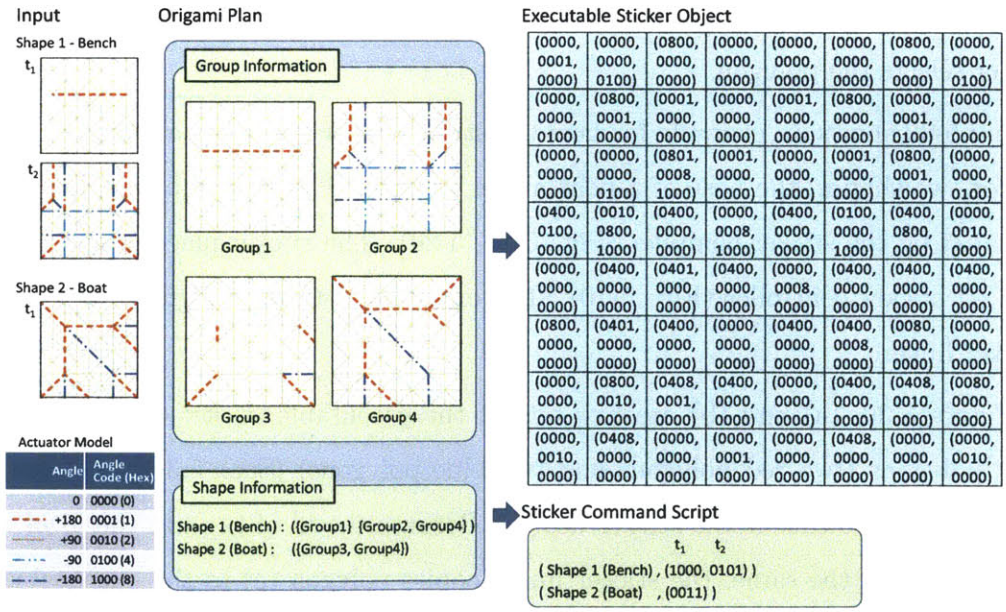


Figure 5-28: Result of the sticker compiler for 8×8 shapes, a bench and a boat. The actuator codes of the executable sticker object is hex. The hex actuator codes of the actuator model are in ( ).

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

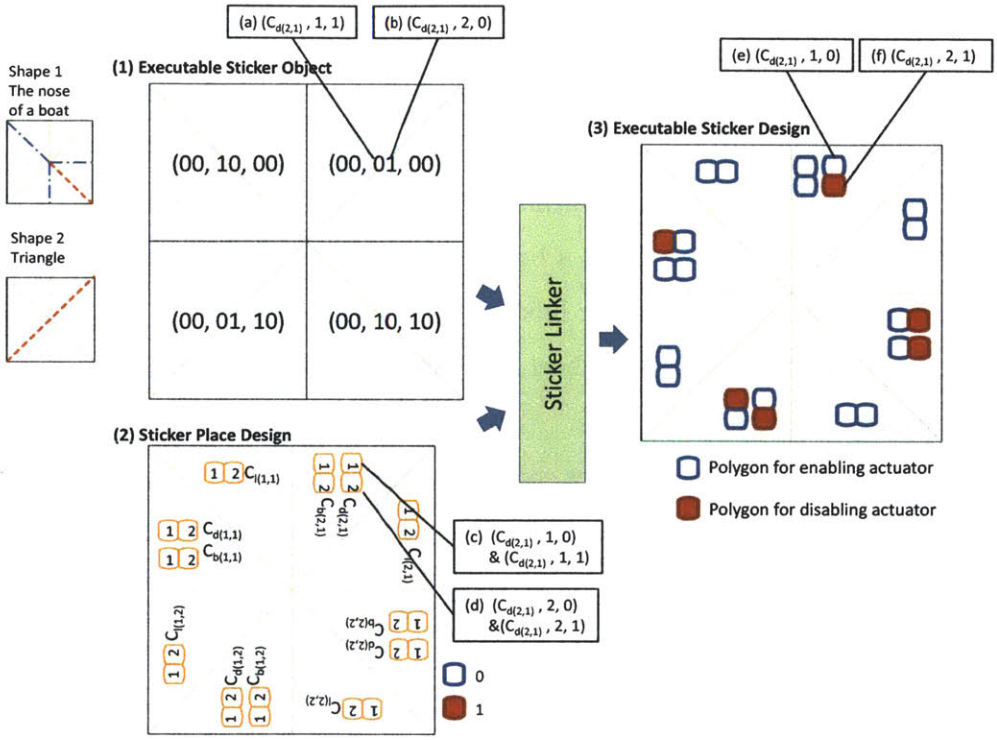


Figure 5-30: Linking an executable sticker design for the nose of boat and triangle(Fig. 5-29)

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.

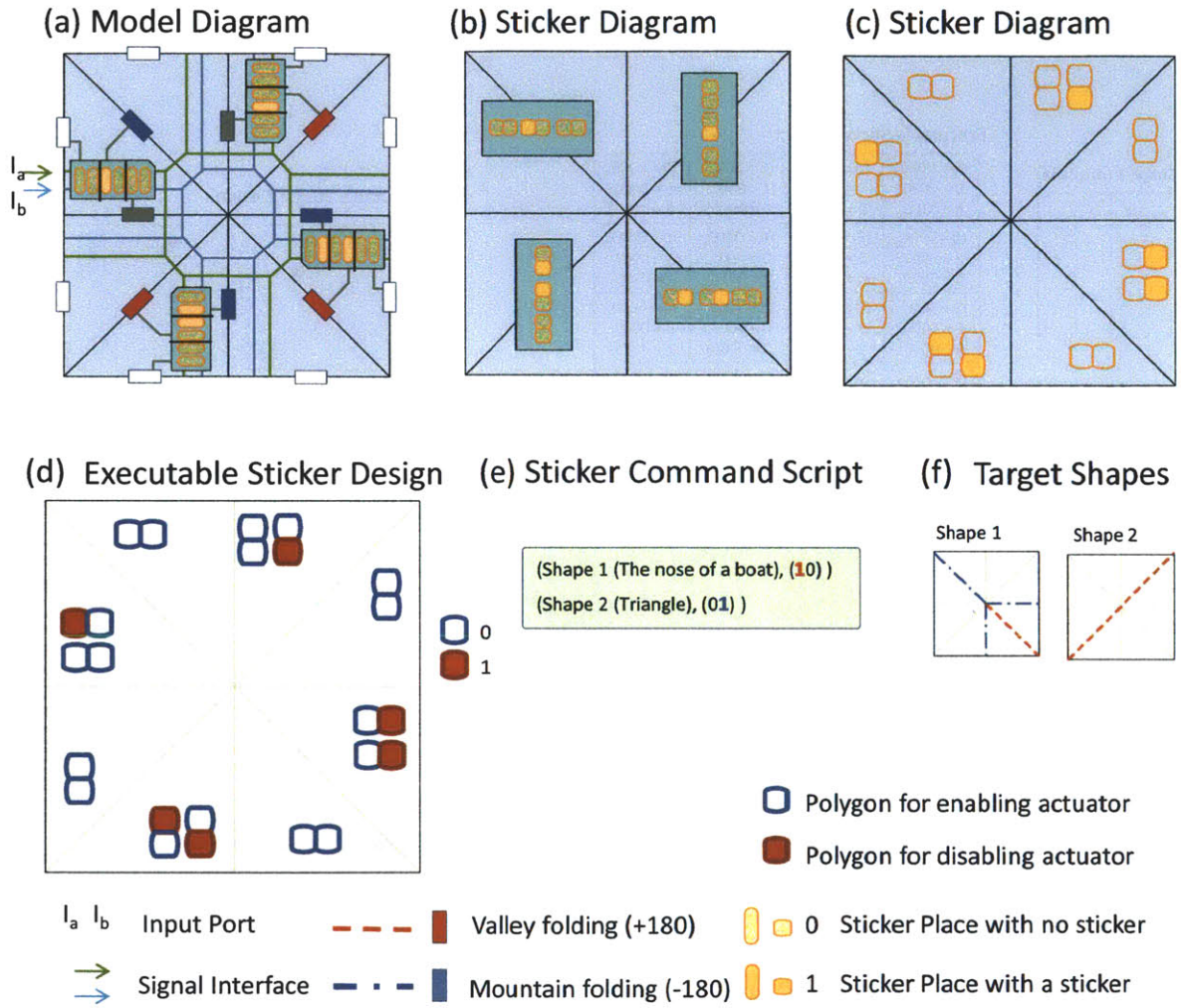


Figure 5-31: Example of a sticker place design for a 2×2 sticker controller for the nose of a boat and a triangle. (a)(b)(c) present a same model for the 2×2 sticker controller in three different diagrams. (d) presents the executable sticker design. The model contains the executable sticker according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

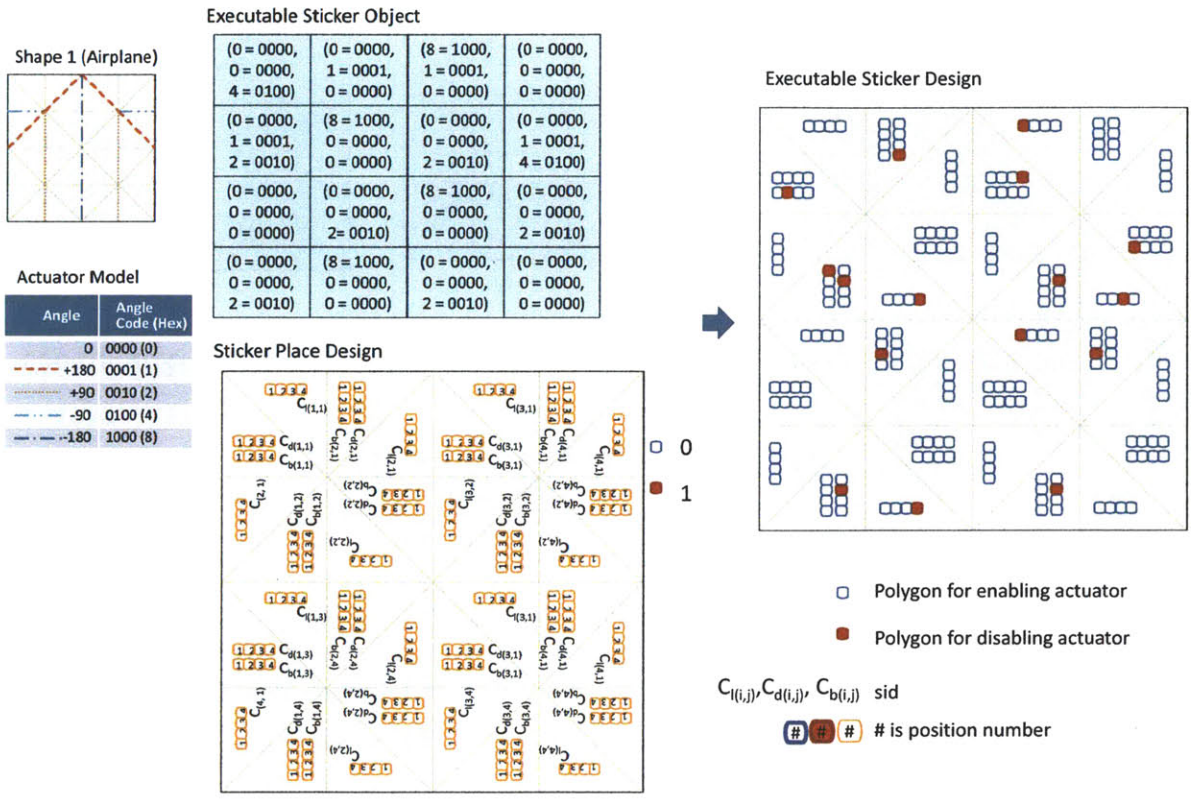


Figure 5-32: Result of the sticker linker for a single origami shape, an airplane. Given the executable sticker object (Fig. 5-26) and the 4×4 3-1-4 sticker space design (Sec. 5.3.1), the sticker linker generates the executable sticker design. The actuator codes of the executable sticker object show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

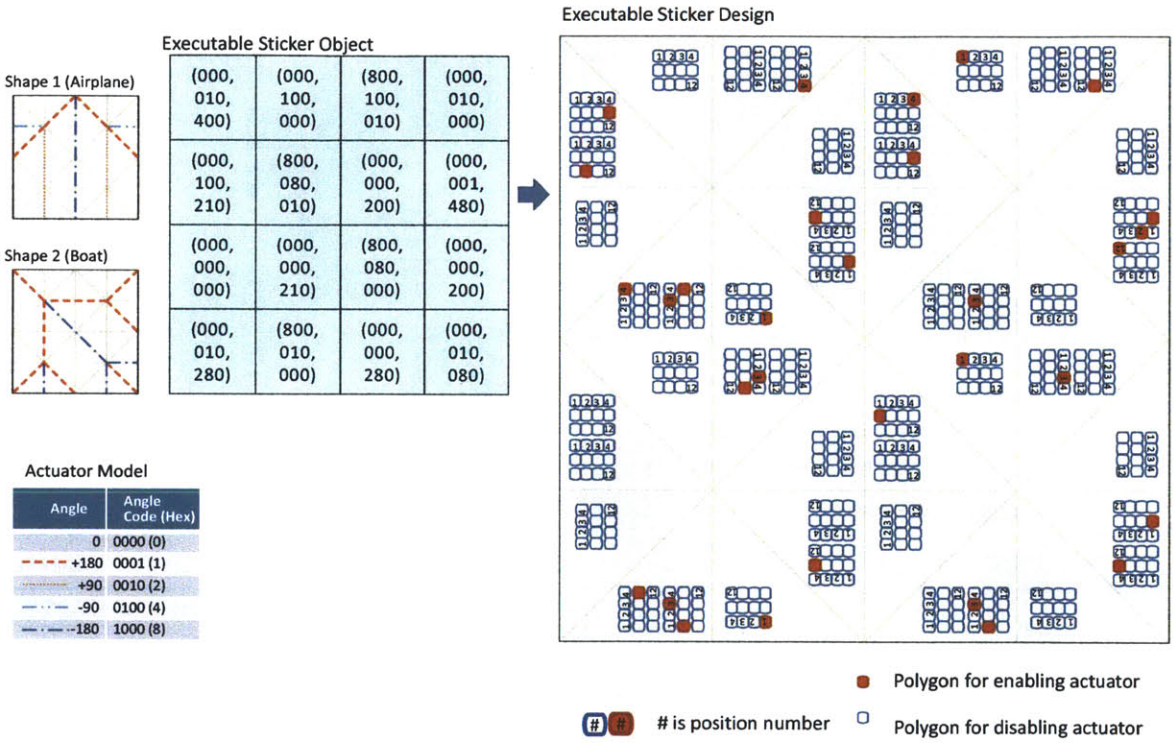


Figure 5-33: Result of the sticker linker for two origami shape, an airplane and a boat. Given the executable sticker object (Fig. 5-21) and the 4×4 3-3-4 sticker space design (Sec. 5.3.1), the sticker linker generates the executable sticker design. The actuator codes of the executable sticker object 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.

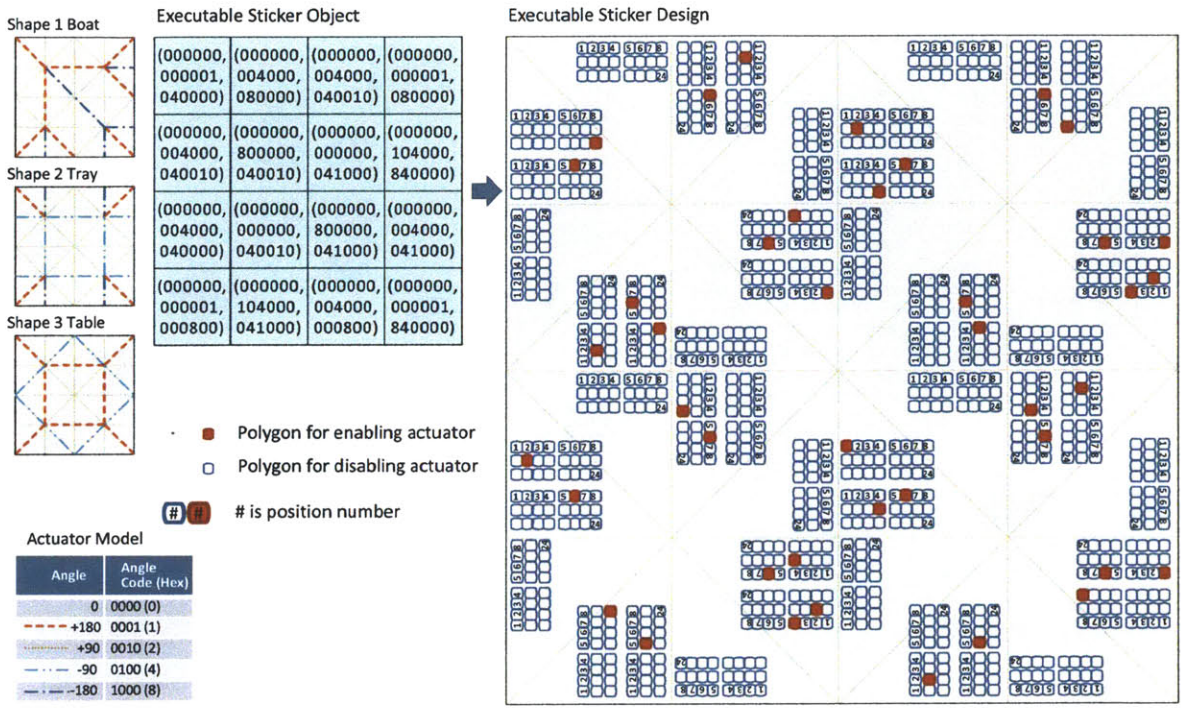


Figure 5-34: Result of the sticker linker for three origami shape, a boat, a tray, and a table. Given the executable sticker object (Fig. 5-27) and the 4×4 3-6-4 sticker space design (Sec. 5.3.1), the sticker linker generates the executable sticker design. The actuator codes of the executable sticker 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.

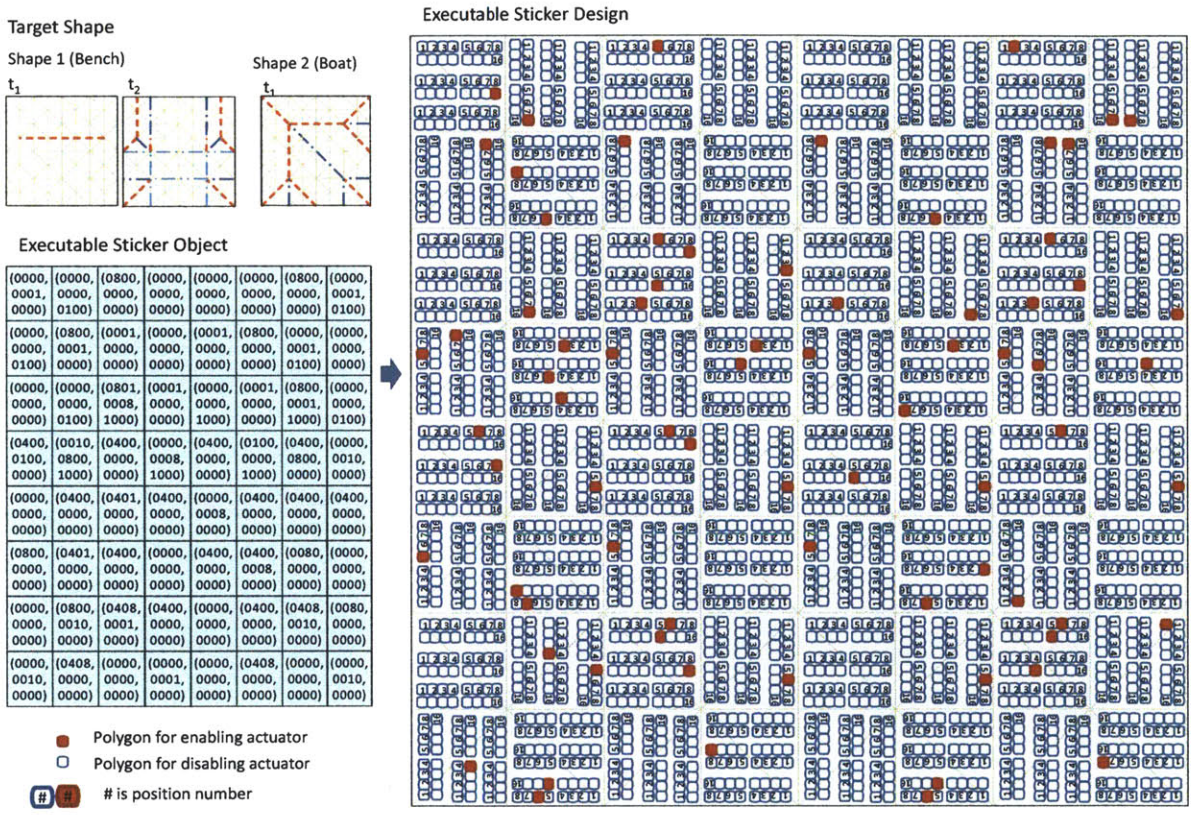


Figure 5-35: Result of the sticker linker for 8×8 origami shapes, a bench and a boat. Given the executable sticker object (Fig. 5-27) and the 8×8 3-4-4 sticker space design (Sec. 5.3.1), the sticker linker generates the executable sticker design. The actuator codes in the executable sticker object are 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.

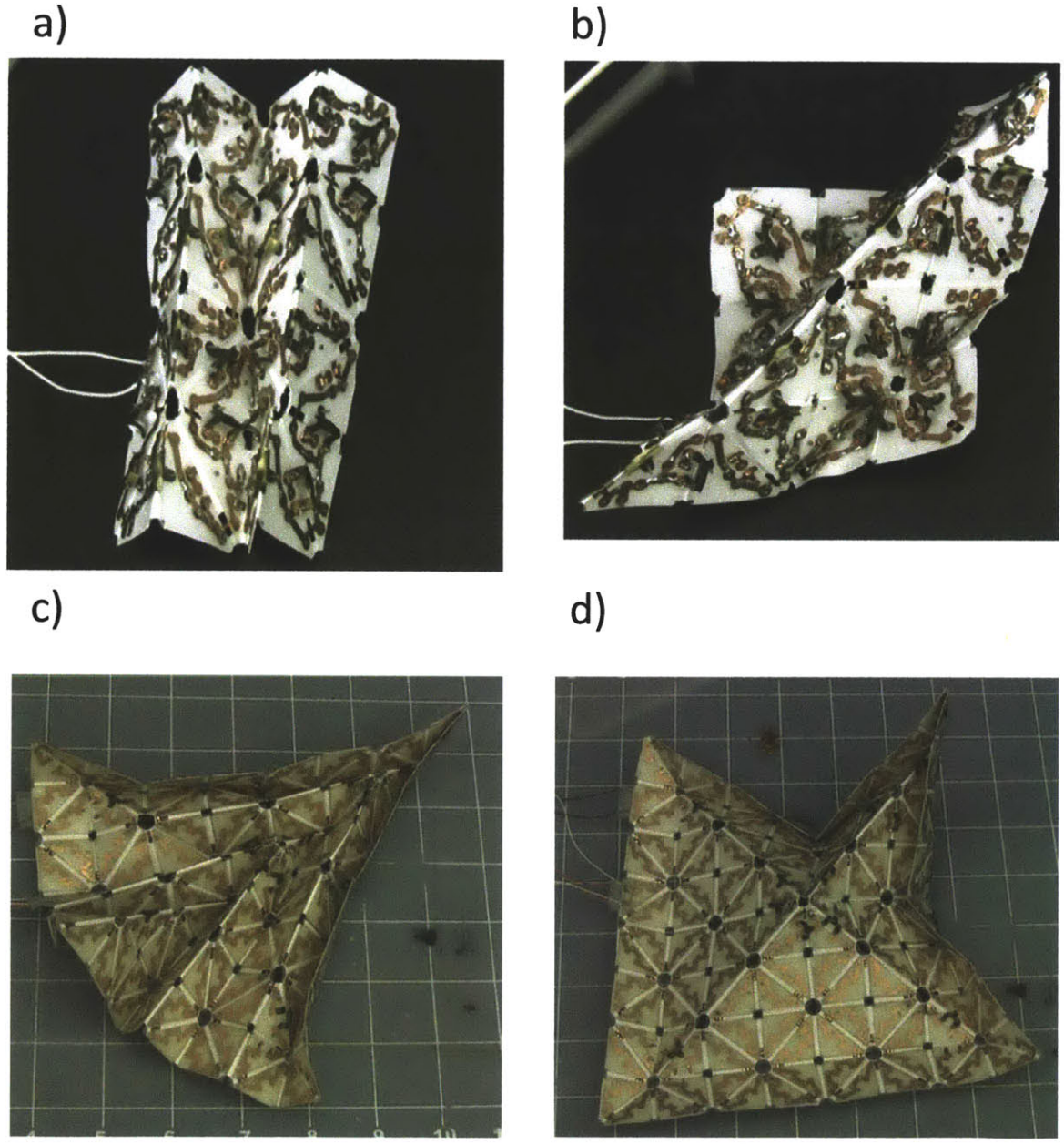


Figure 6-1: 4 Examples of self-folding sheets. (a) Vertical Folding (b) Diagonal Folding (c) Space Shuttle (d) Hat. The objects in (a) and (b) were folded from the same 4×4 sheet. The objects in (c) and (d) were folded from the same 8×8 sheet. (a)(b) 4×4 Self-folding sheet (c)(d) 8×8 Self-Folding Sheet

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

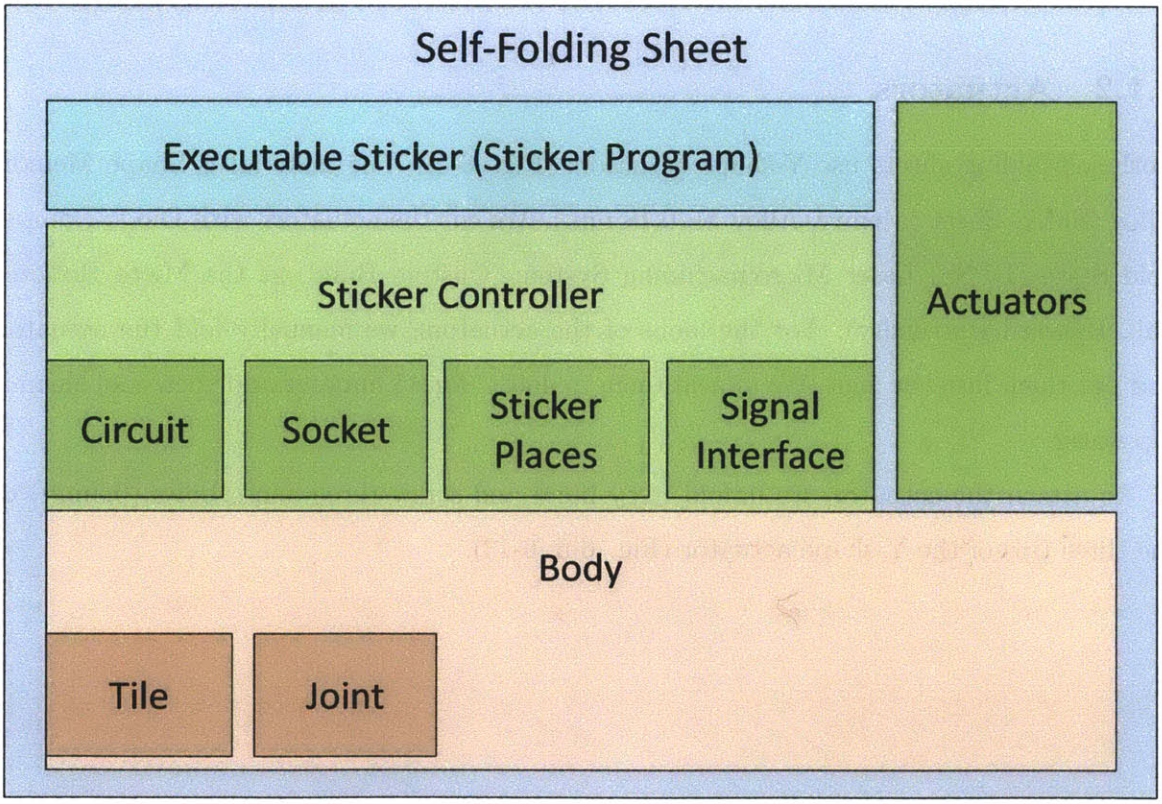


Figure 6-2: Architecture of self-folding sheet

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).

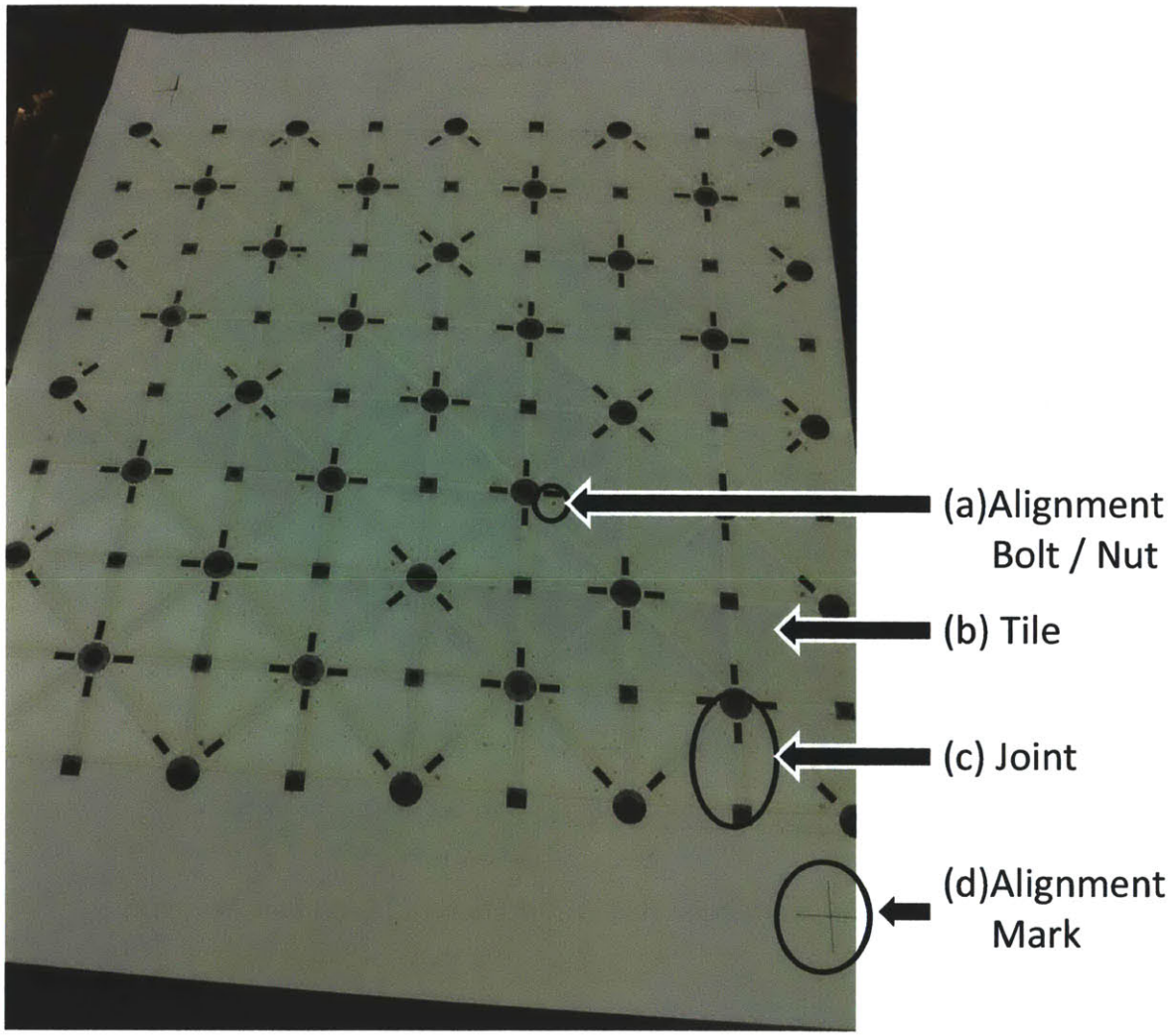


Figure 6-3: Lamination tile and paper joint body (LP Body)

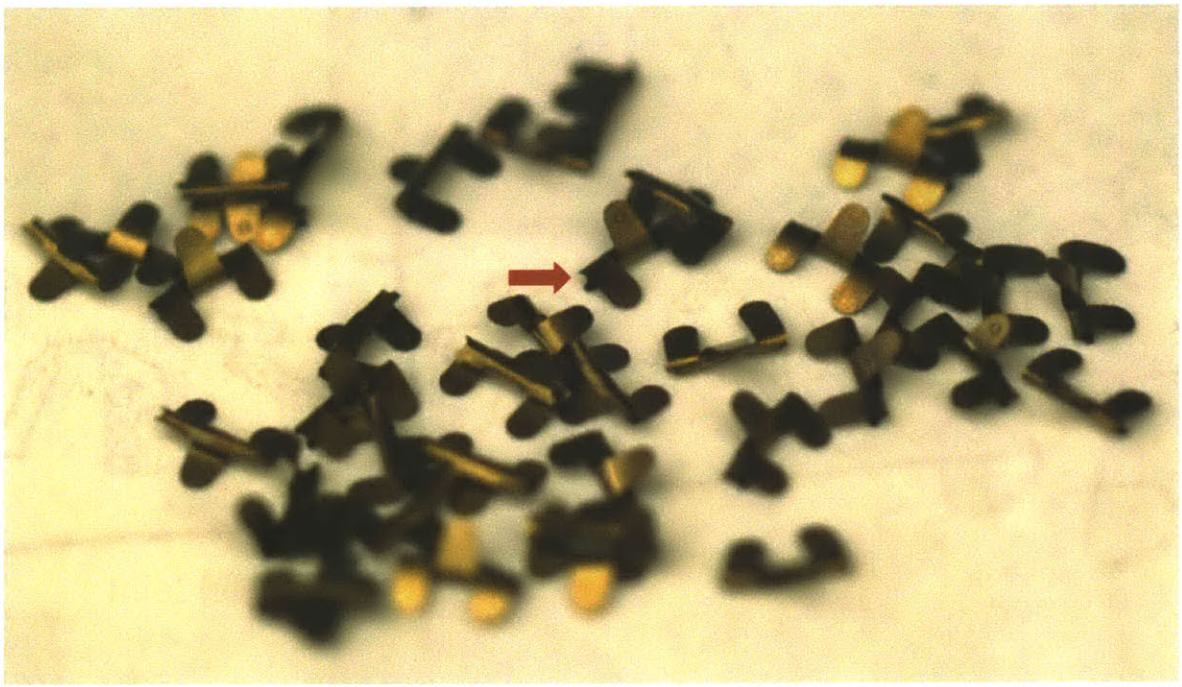


Figure 6-4: Y-Type Actuators. Red Arrow Points a Loop of an Actuator.

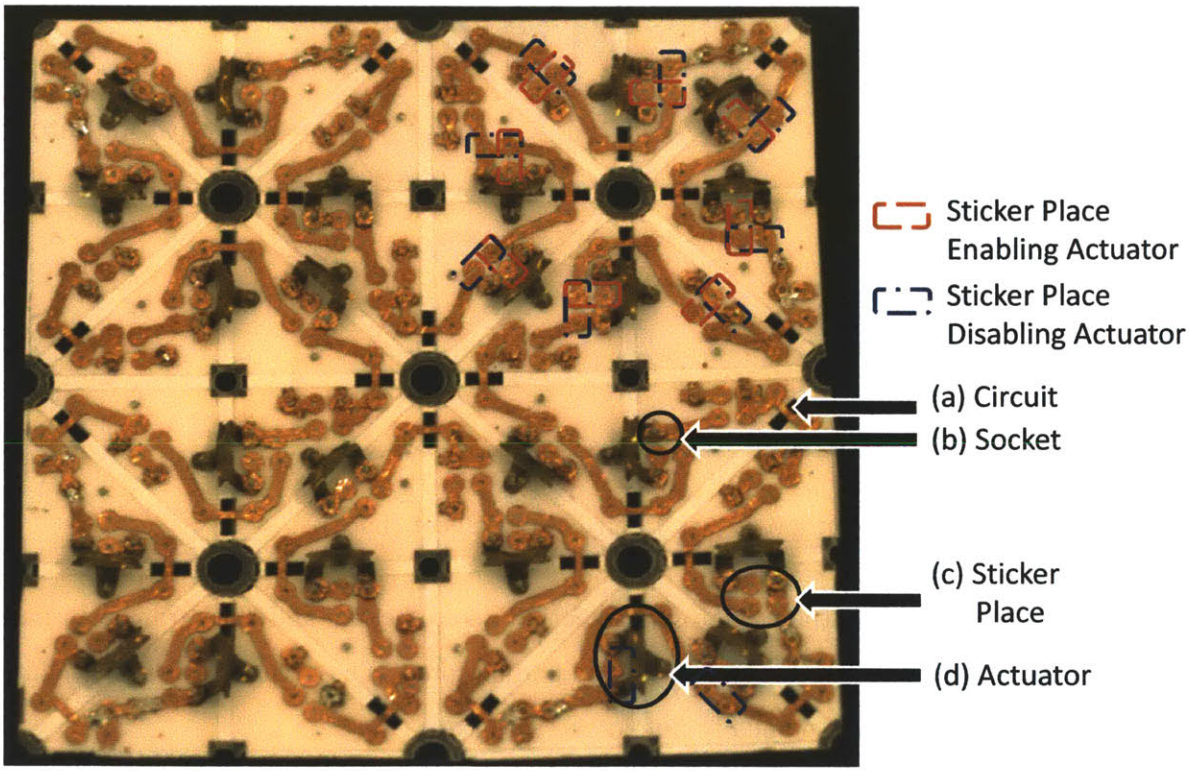


Figure 6-5: 4×4 Self-Folding Sheet without stickers

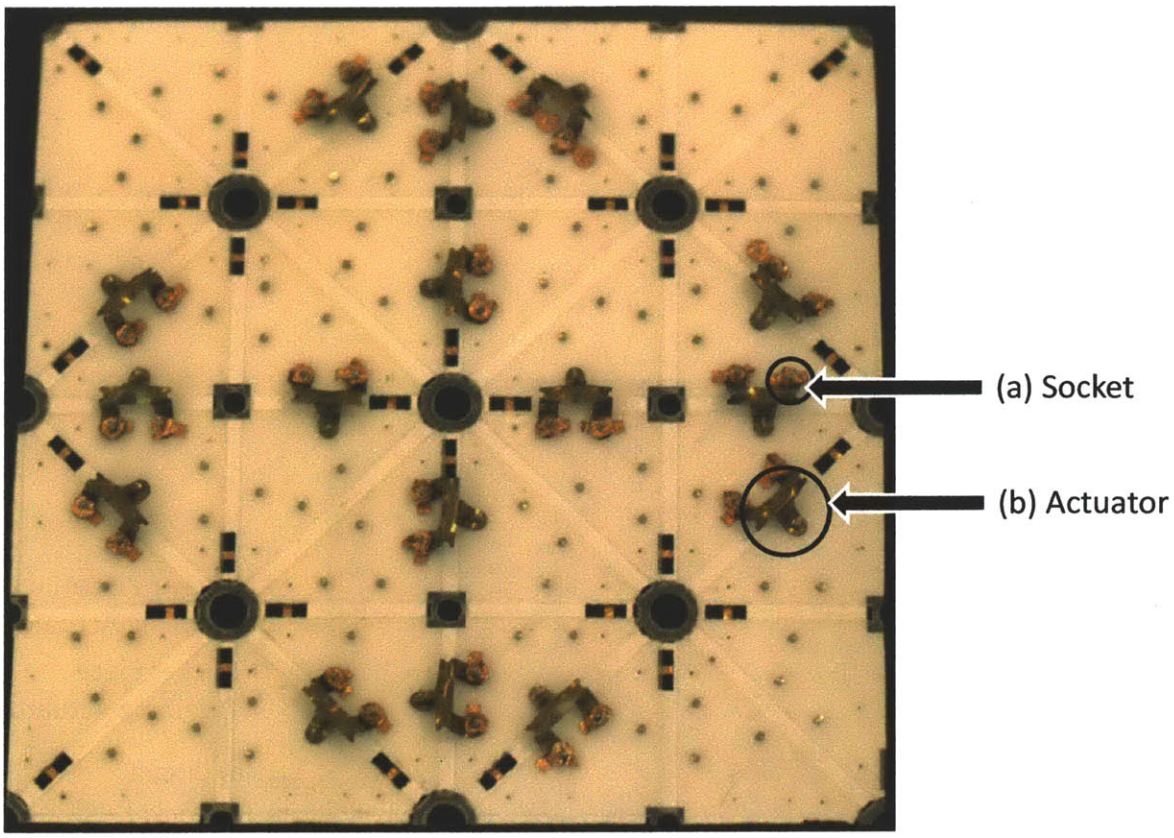


Figure 6-6: 4×4 Self-Folding Sheet (back)

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).

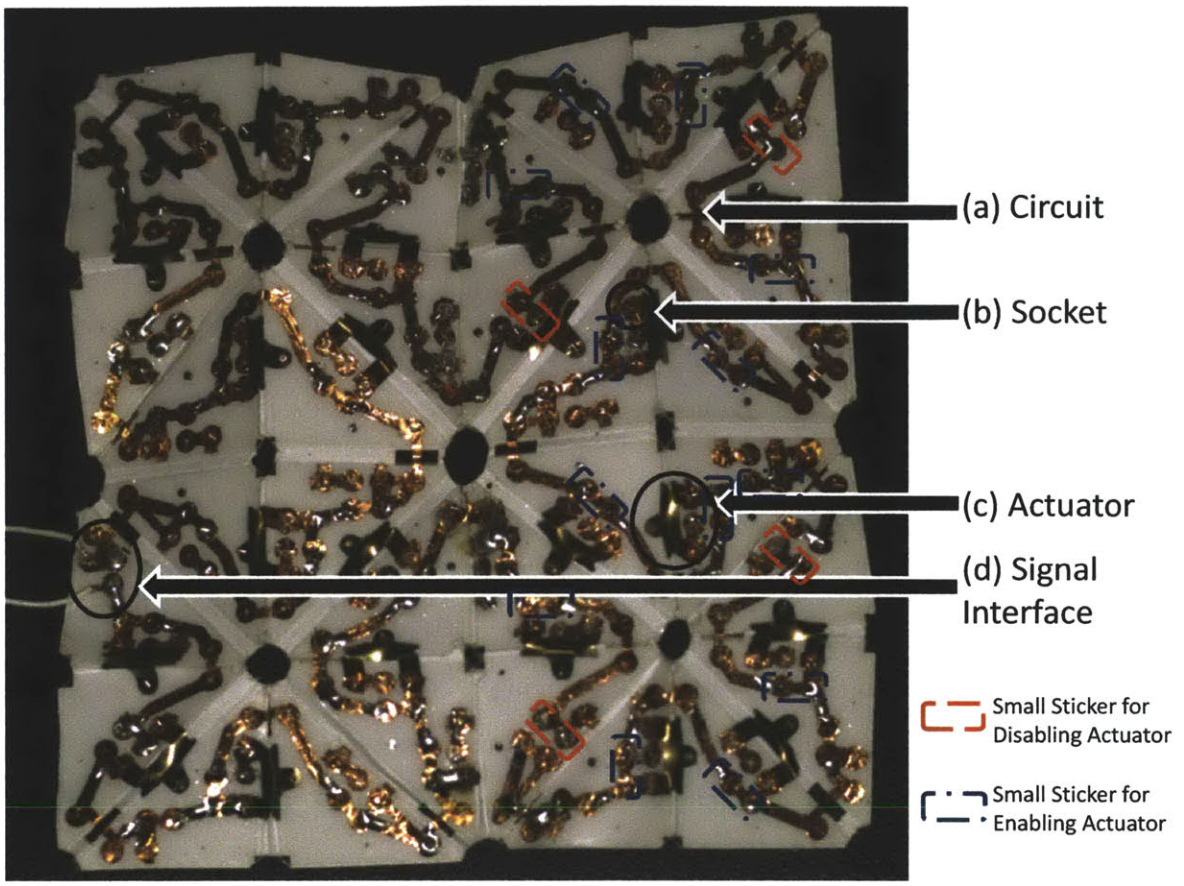


Figure 6-7: 4×4 Self-Folding Sheet with the Executable Sticker (Diagonal Folding Program) (Front)

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.

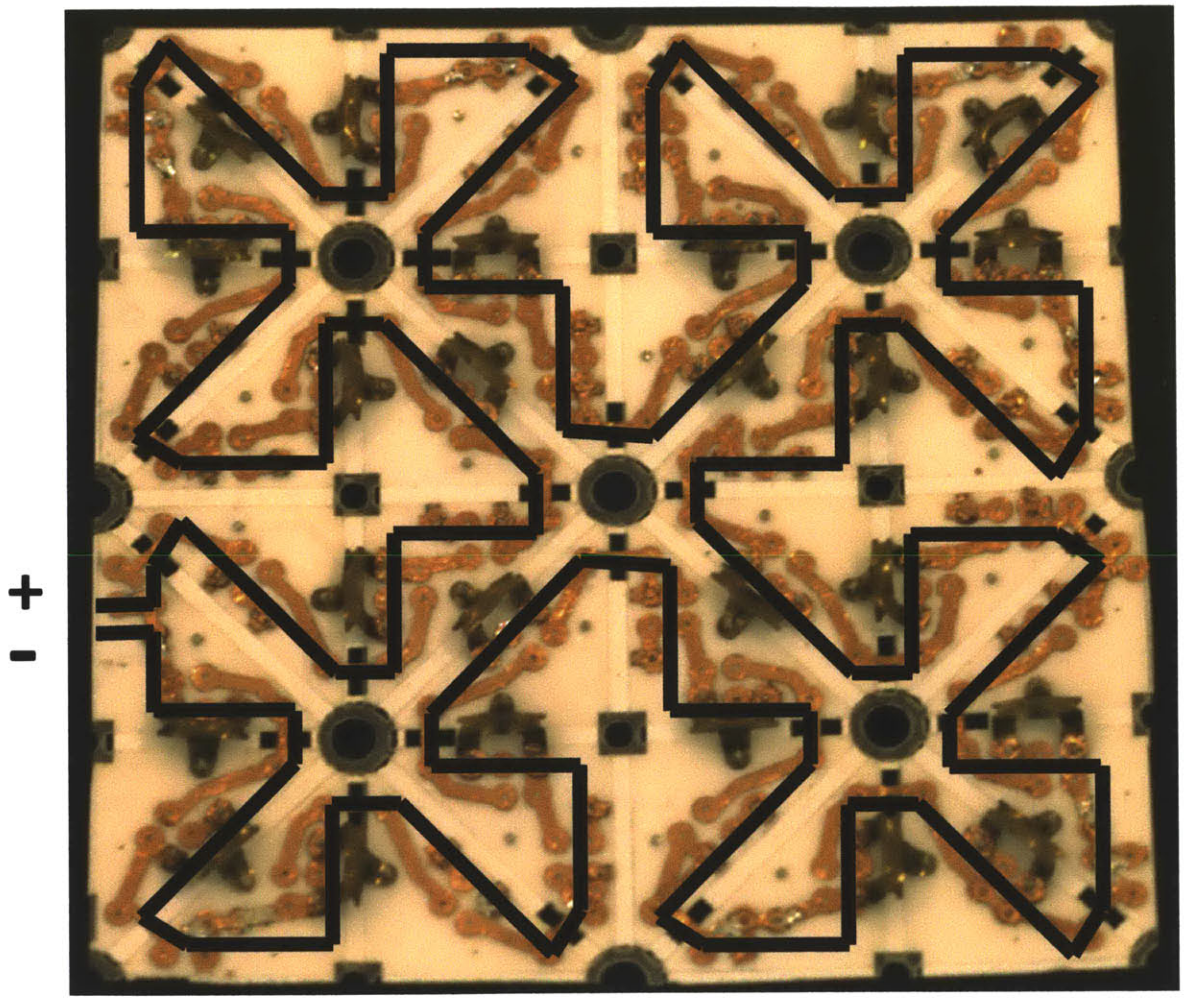


Figure 6-9: The circuit for 4×4 Self-Folding Sheet (with Example)

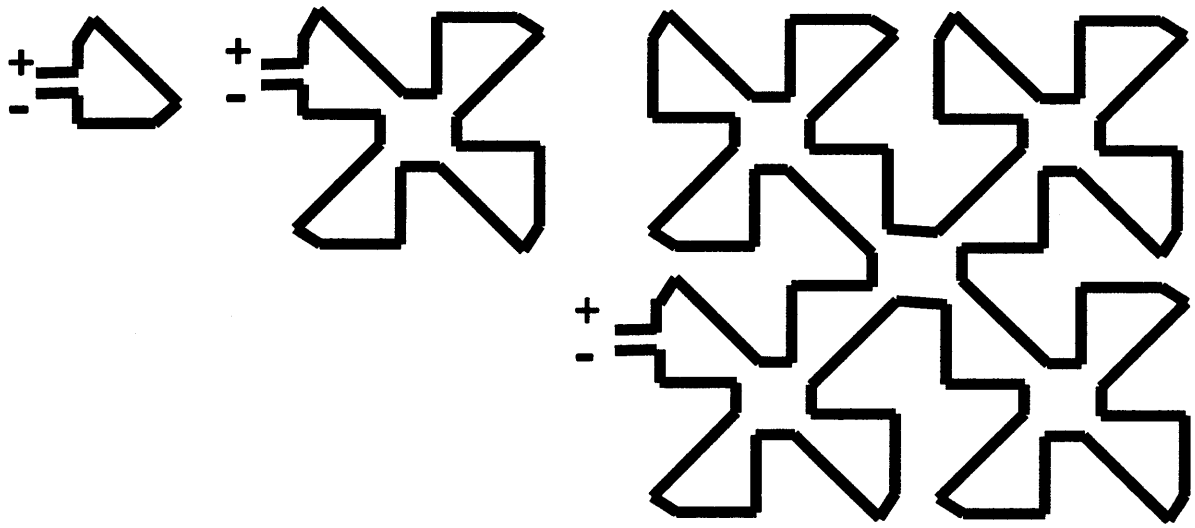


Figure 6-10: The circuits for 1×1, 2×2, and 4×4 Self-Folding Sheets (left, center, right)

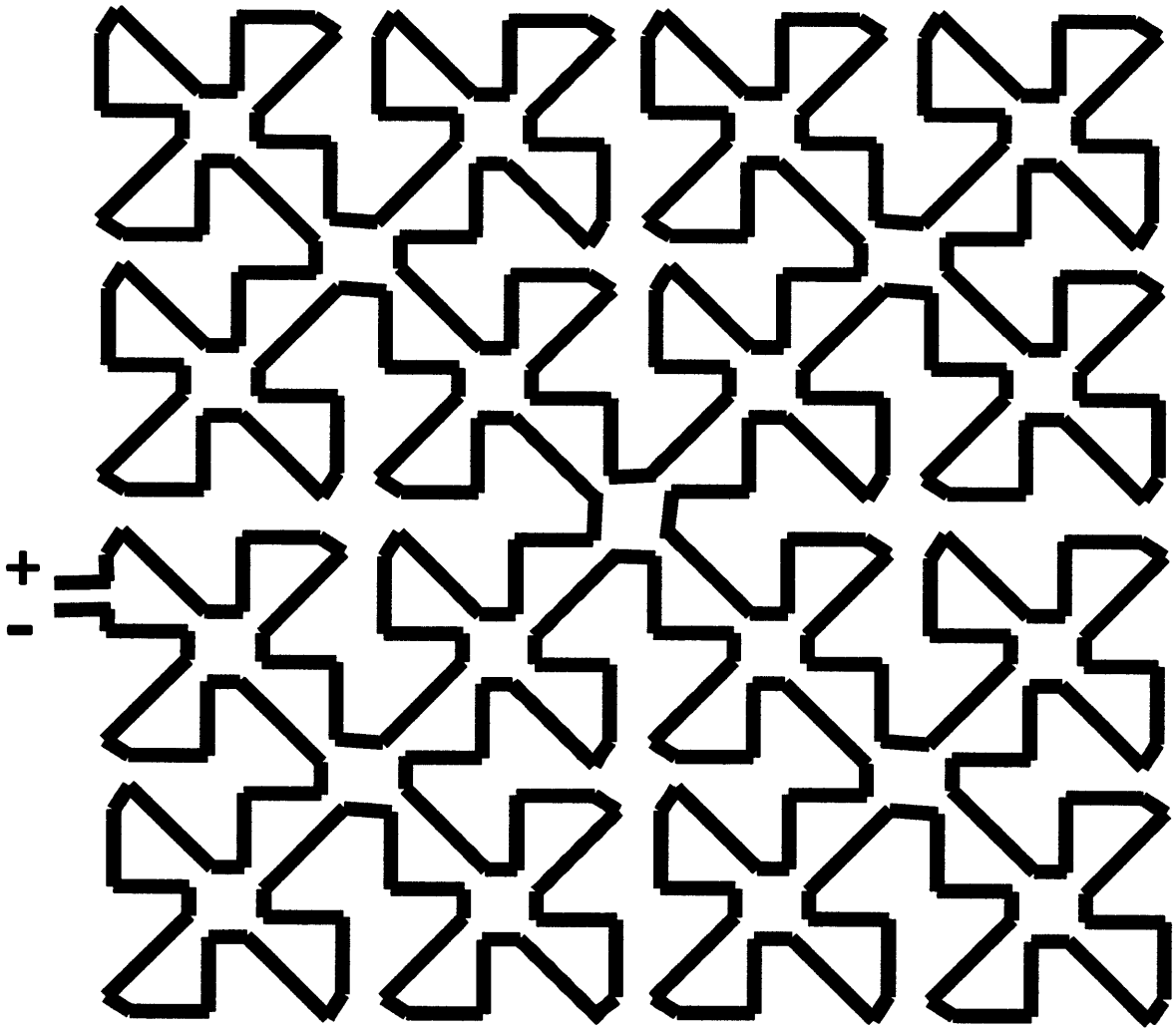


Figure 6-11: The circuit for 8×8 Self-Folding Sheets

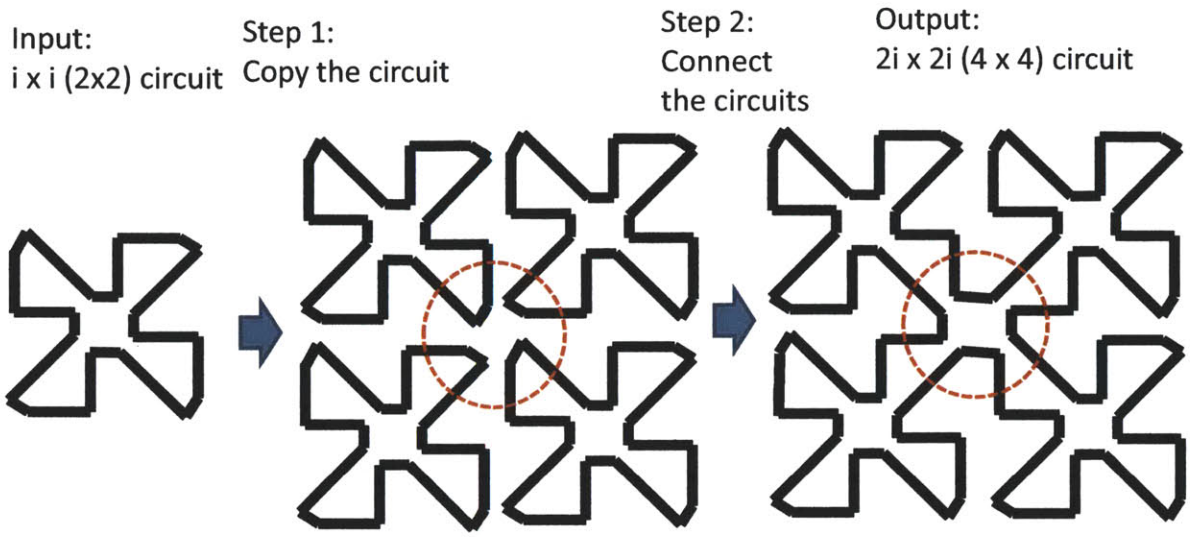


Figure 6-12: The circuit scaling algorithm

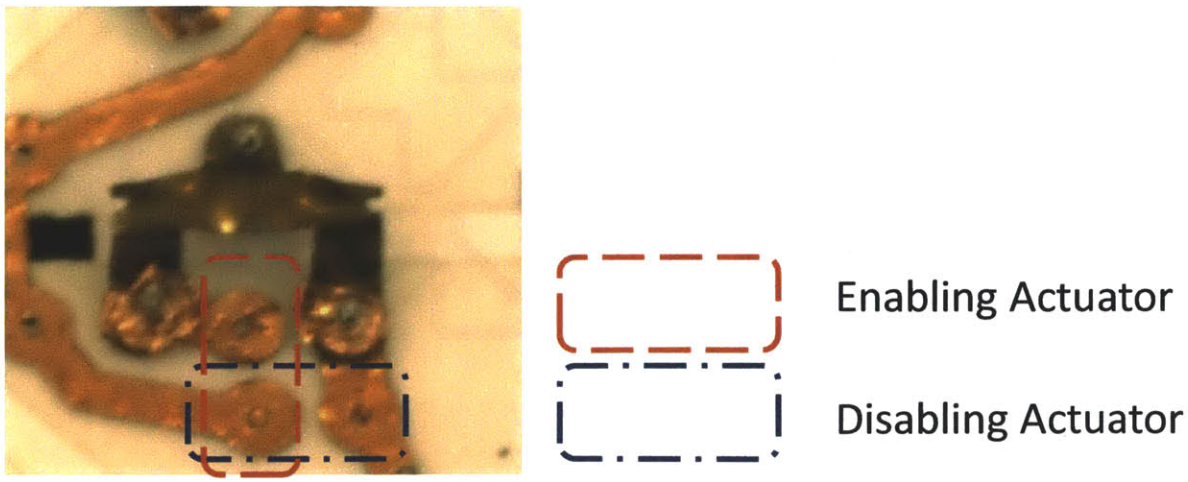


Figure 6-13: Sticker Place of 4×4 Self-Folding Sheet

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.

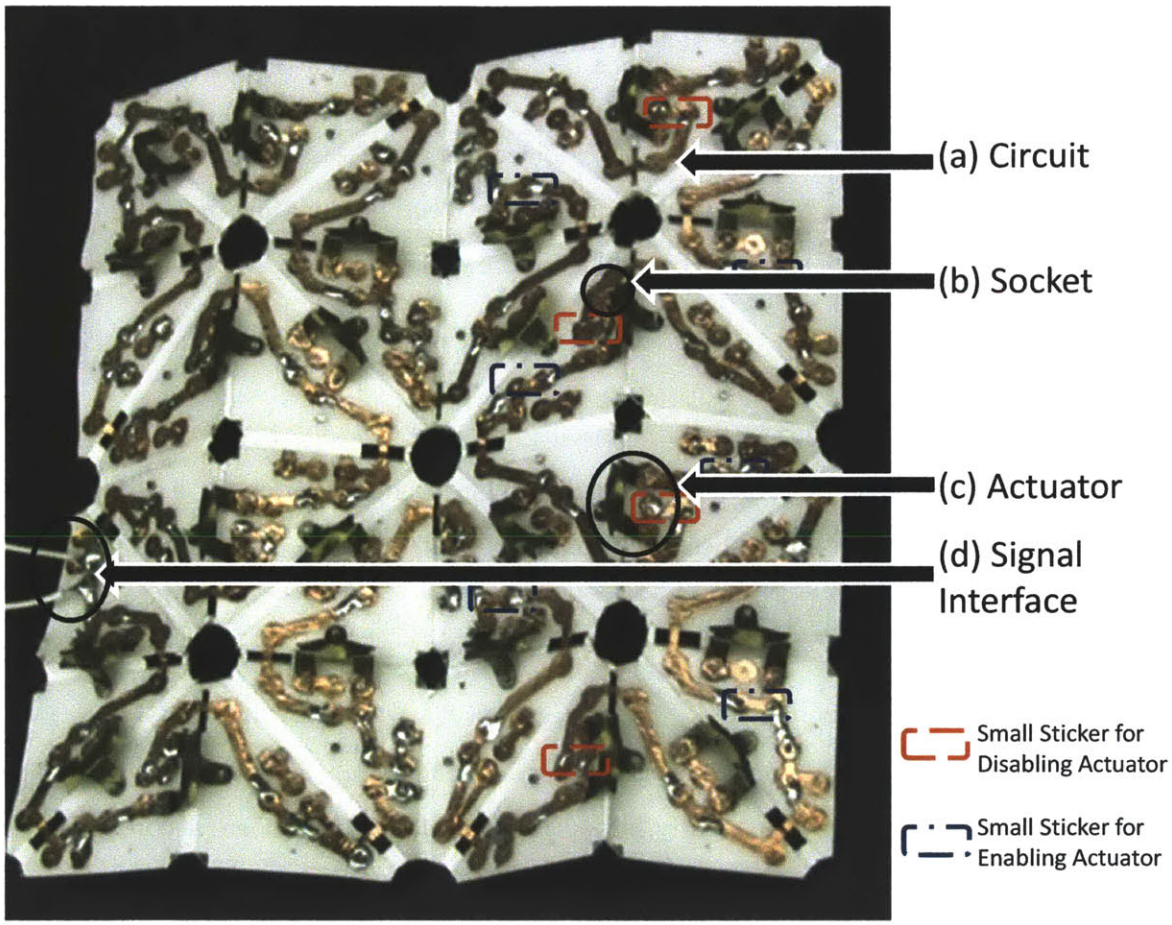


Figure 6-14: 4×4 Self-Folding Sheet with Executable Sticker (Vertical Folding Program).

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

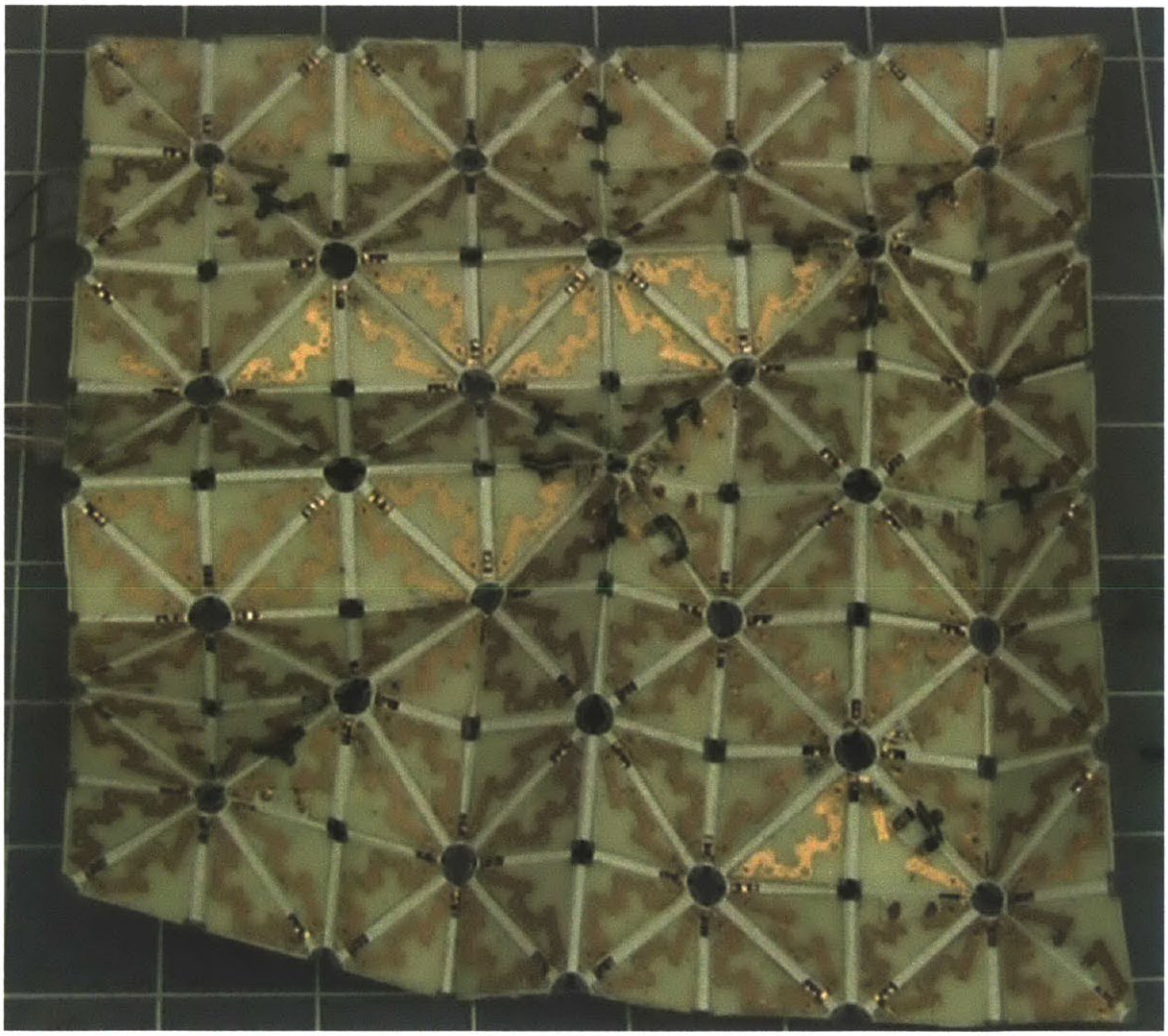


Figure 6-15: 8×8 Self-Folding Sheet

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.

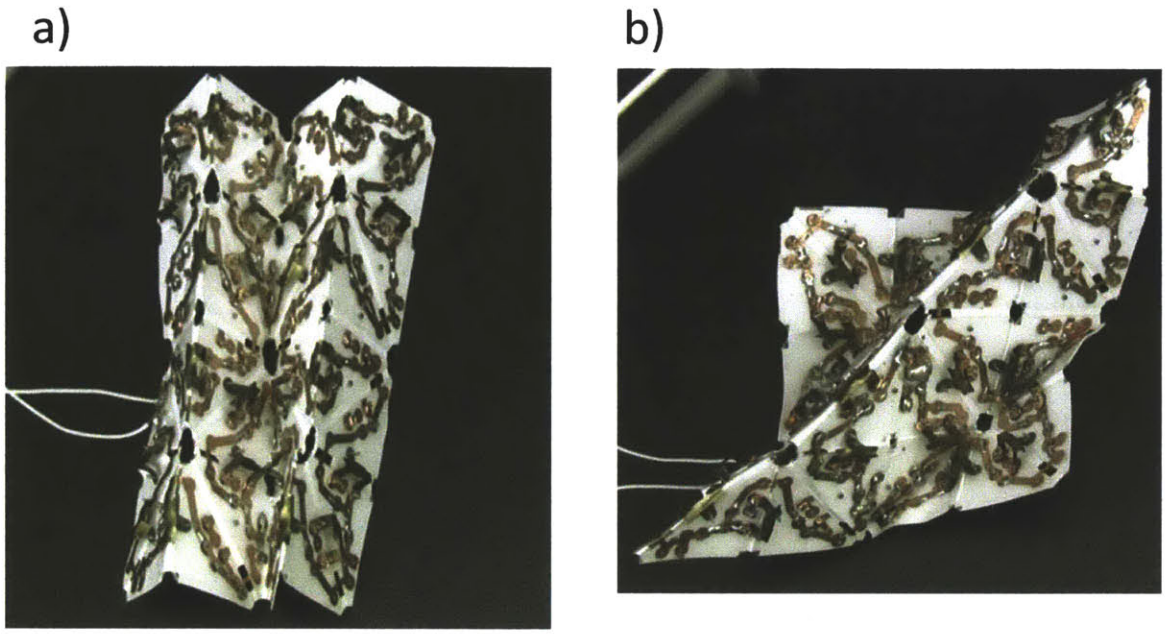


Figure 6-16: Self-Folding Sheets. (a) Vertical Folding (b) Diagonal Folding

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

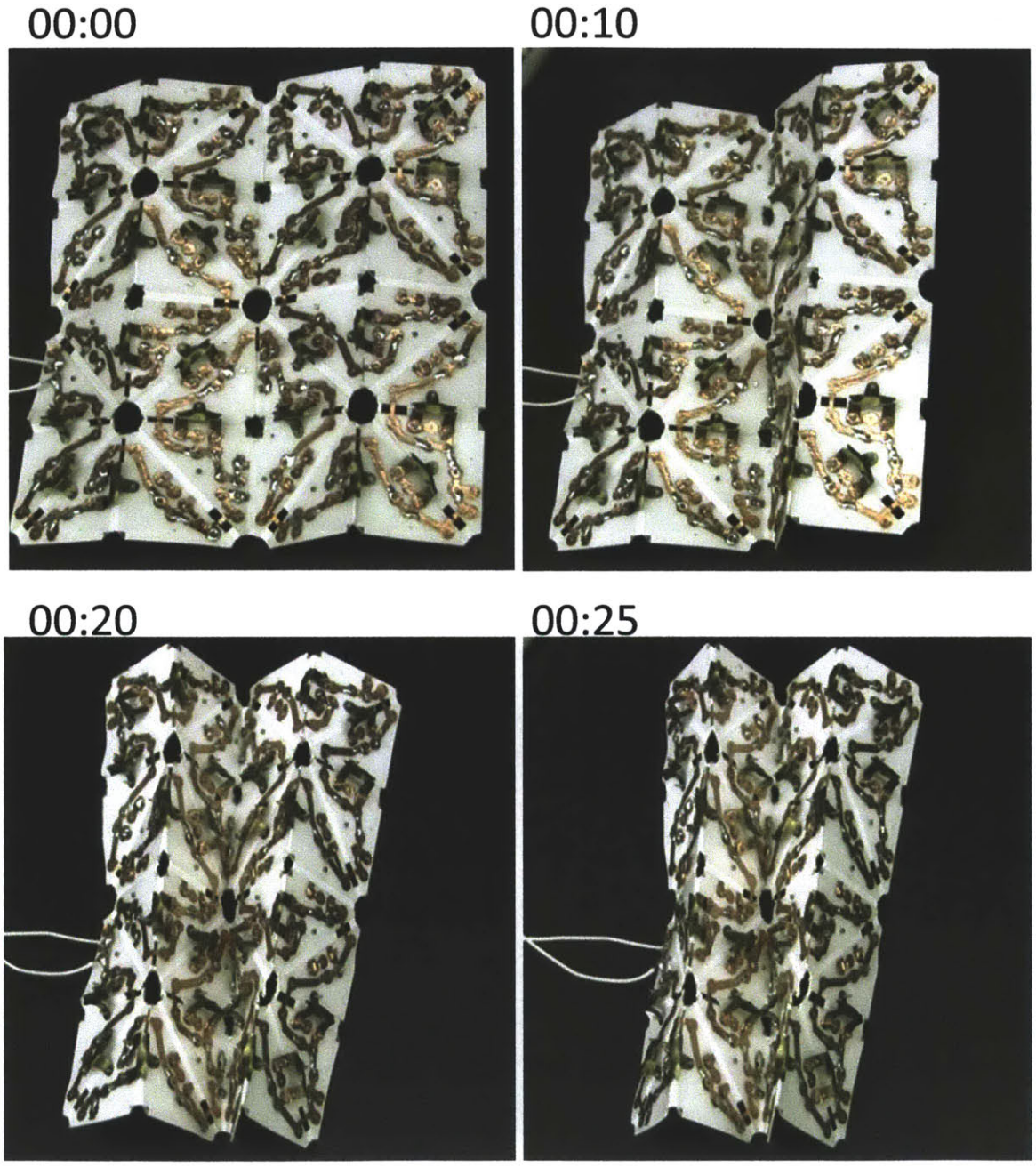


Figure 6-17: Snapshots from controlling the vertical folding 4 × 4 Self-Folding Sheet

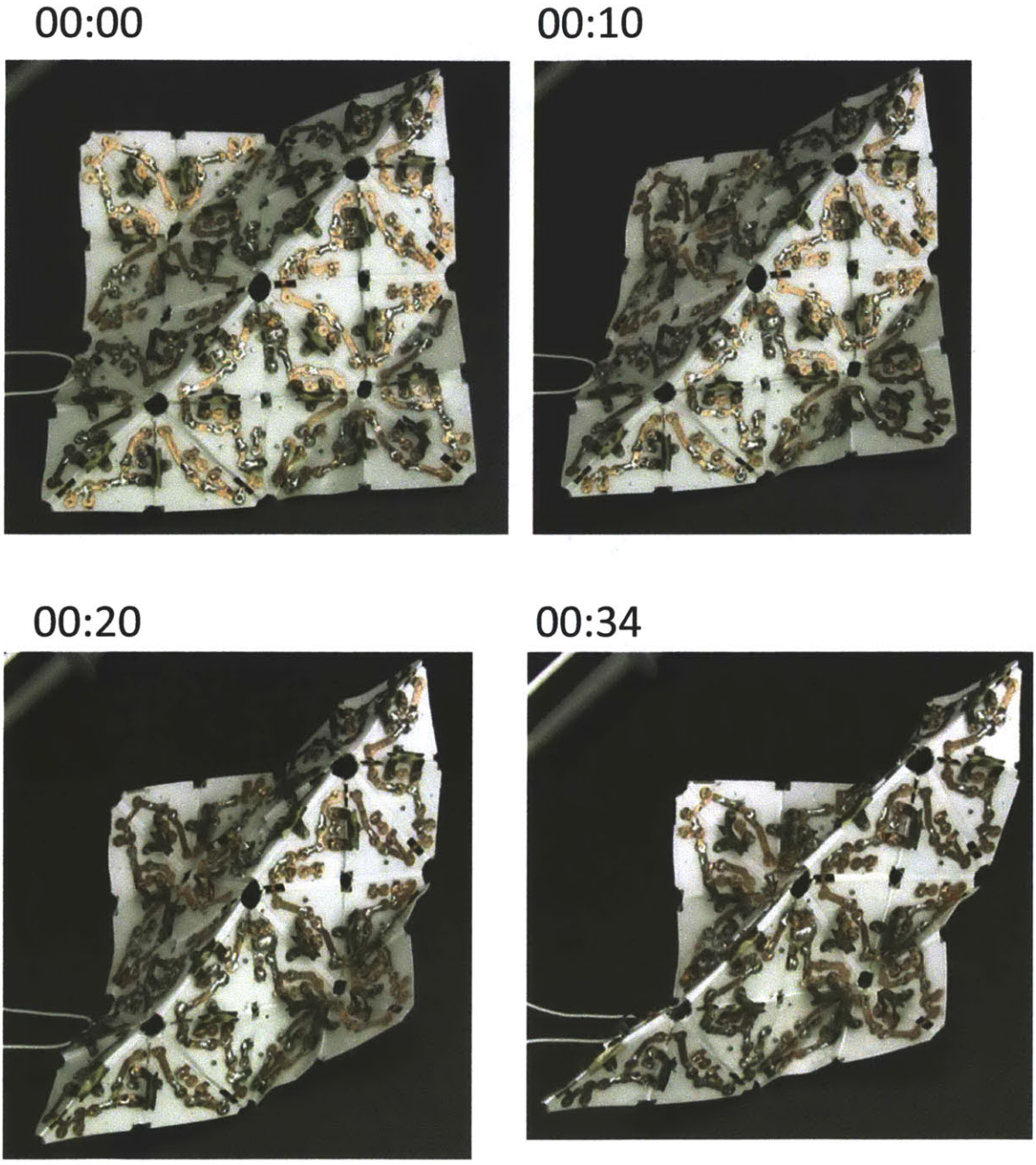


Figure 6-18: Snapshots from controlling the diagonal folding 4×4 Self-Folding Sheet

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.

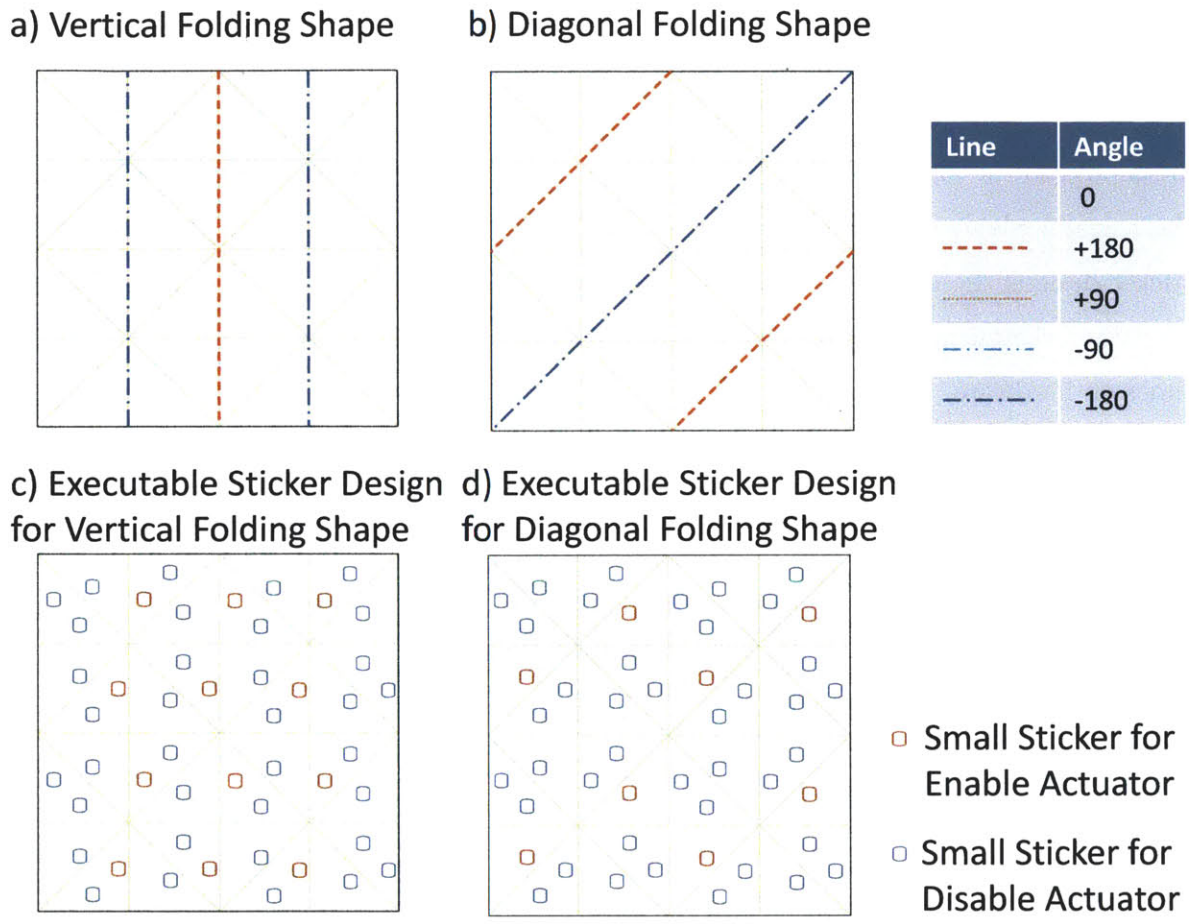


Figure 6-19: Results of the sticker programming algorithm. (a)(b) Origami plan. (c)(d) Executable sticker design. Each small square shows the sticker type for each sticker place. We input the vertical folding program according to (c) as shown in Figure 6-7. We input the diagonal folding program according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

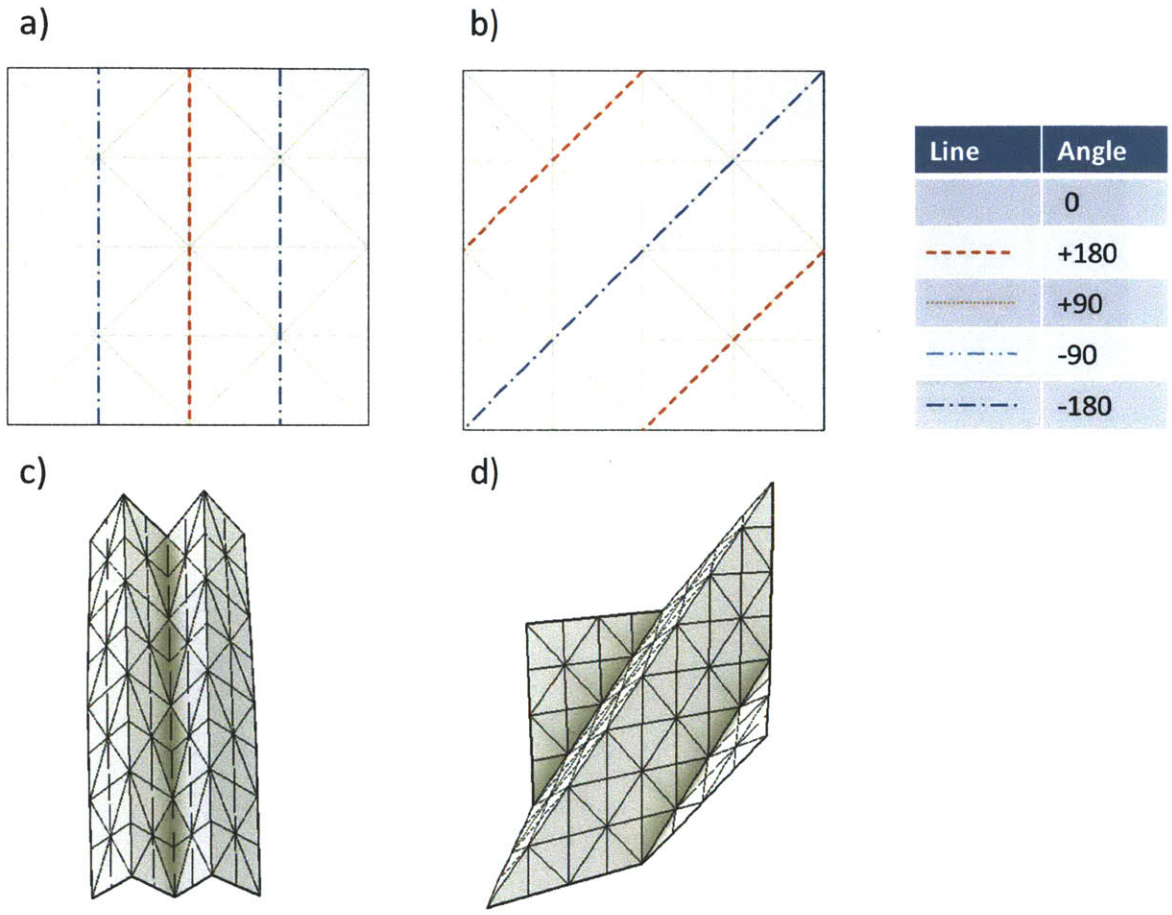


Figure 6-20: Target Shapes for Basic Motion. (a)(c) Vertical Folding (b)(d) Diagonal Folding

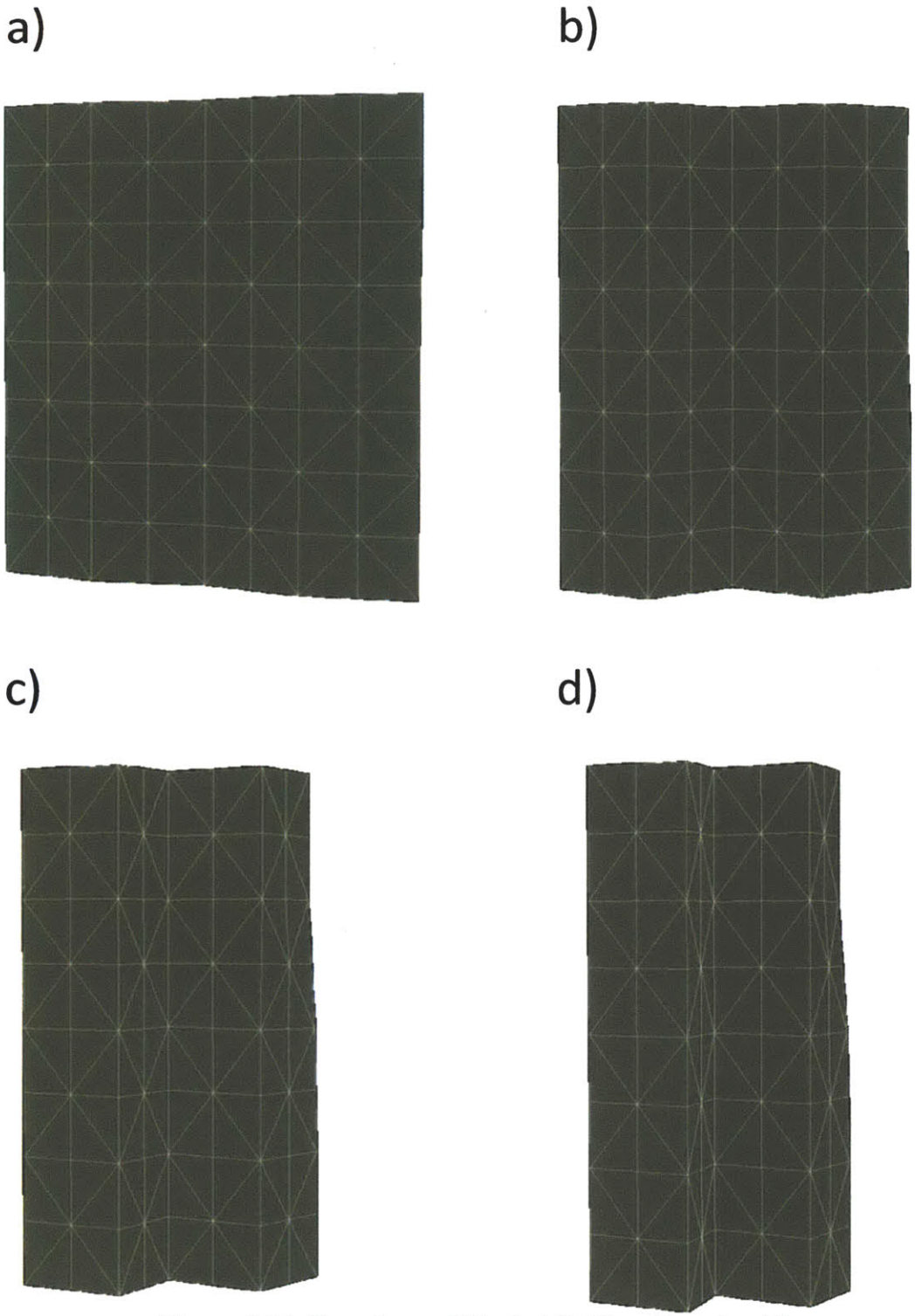


Figure 6-21: Snapshots of Vertical Folding Planning [2]

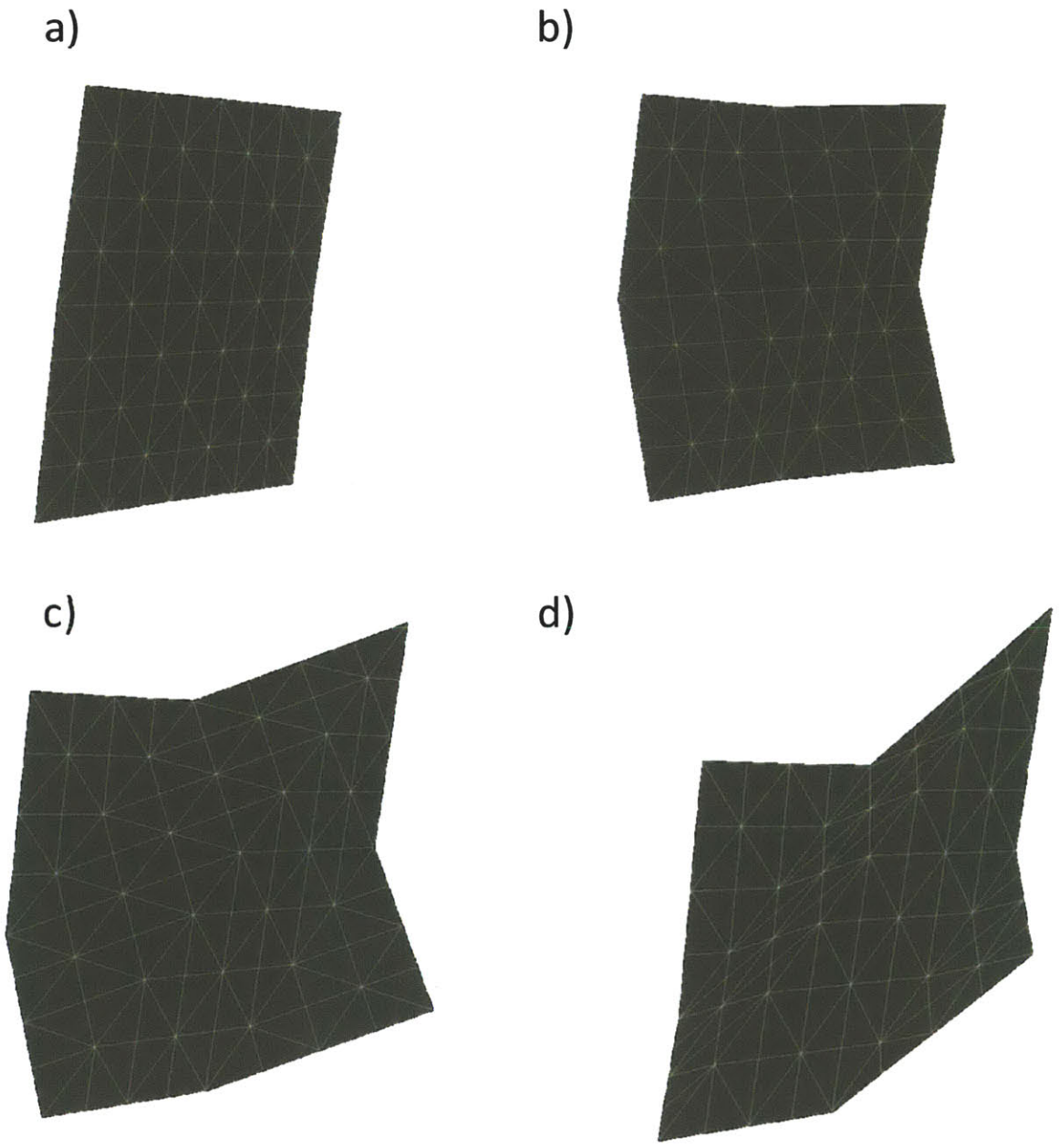


Figure 6-22: Snapshots of Diagonal Folding Planning [2]

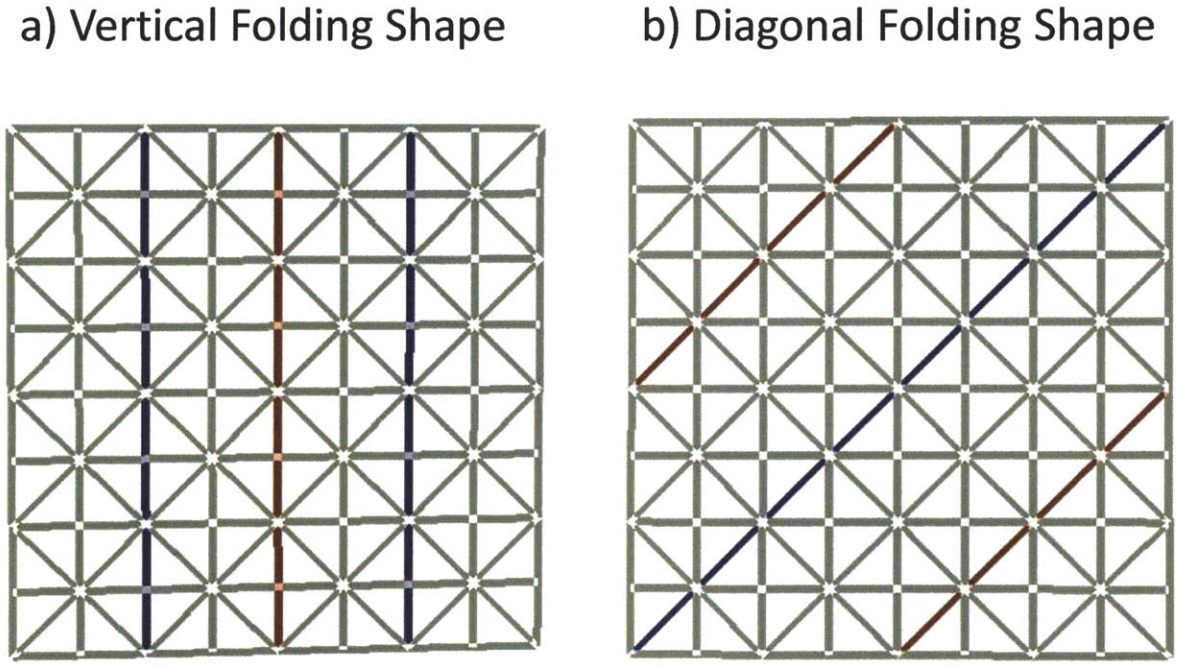


Figure 6-23: Snapshot of Sticker Program Planing. The red line denotes +180° folding. The blue line denotes -180° folding.

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 Angles [1]	Target Angles	Folding Achievement (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 Disabled Actuators	Folding Actuators	Disabled Actuators / Folding Actuators	Disabled 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.

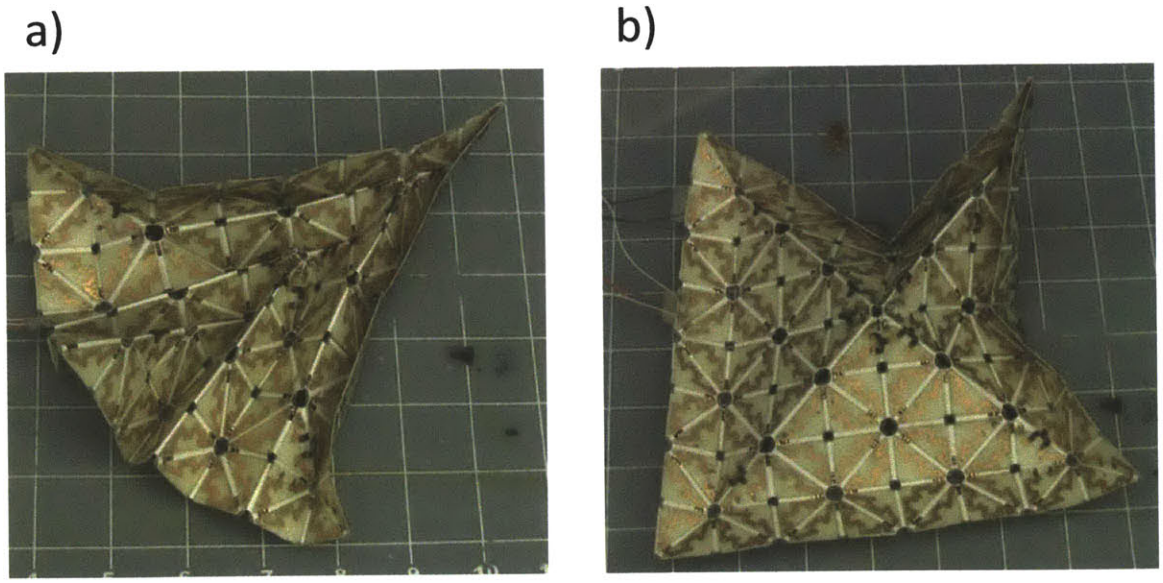


Figure 6-24: Two 8×8 self-folding sheet examples at the end of the self-folding operation. (a) Space Shuttle (b) Hat

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.

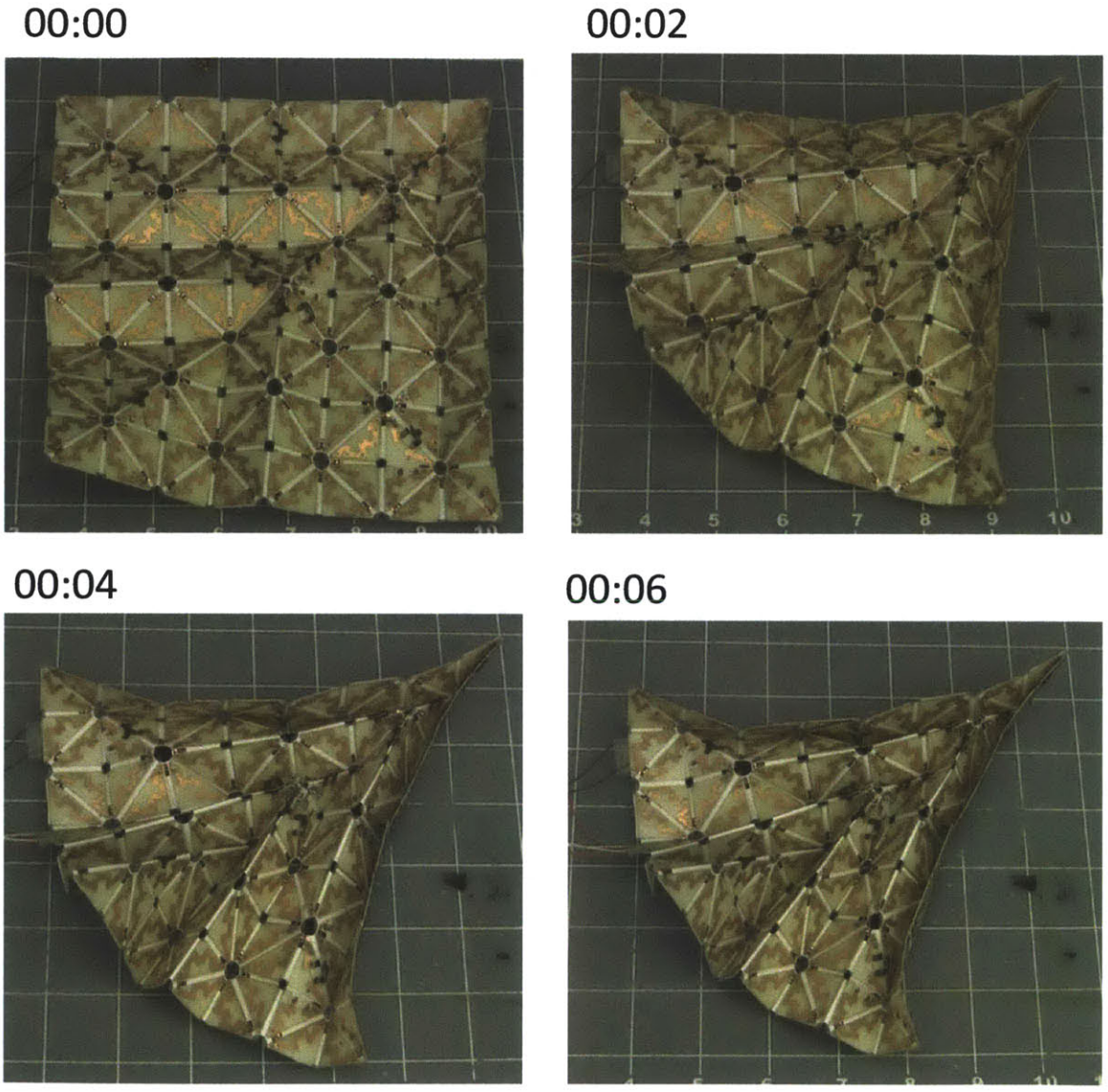


Figure 6-25: Space Shuttle: 8×8 Self-Folding Sheet

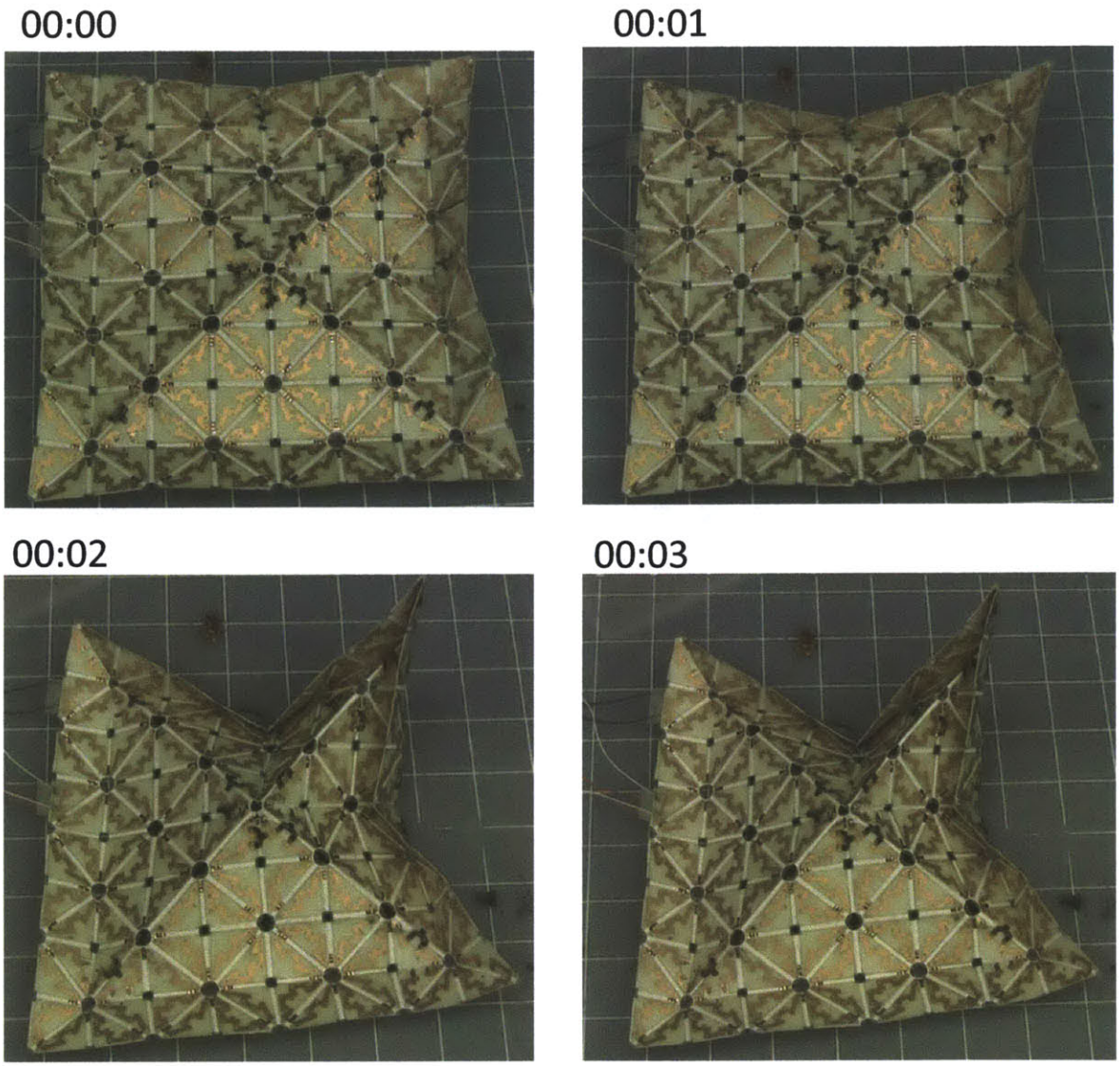


Figure 6-26: Hat: 8×8 Self-Folding Sheet

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.

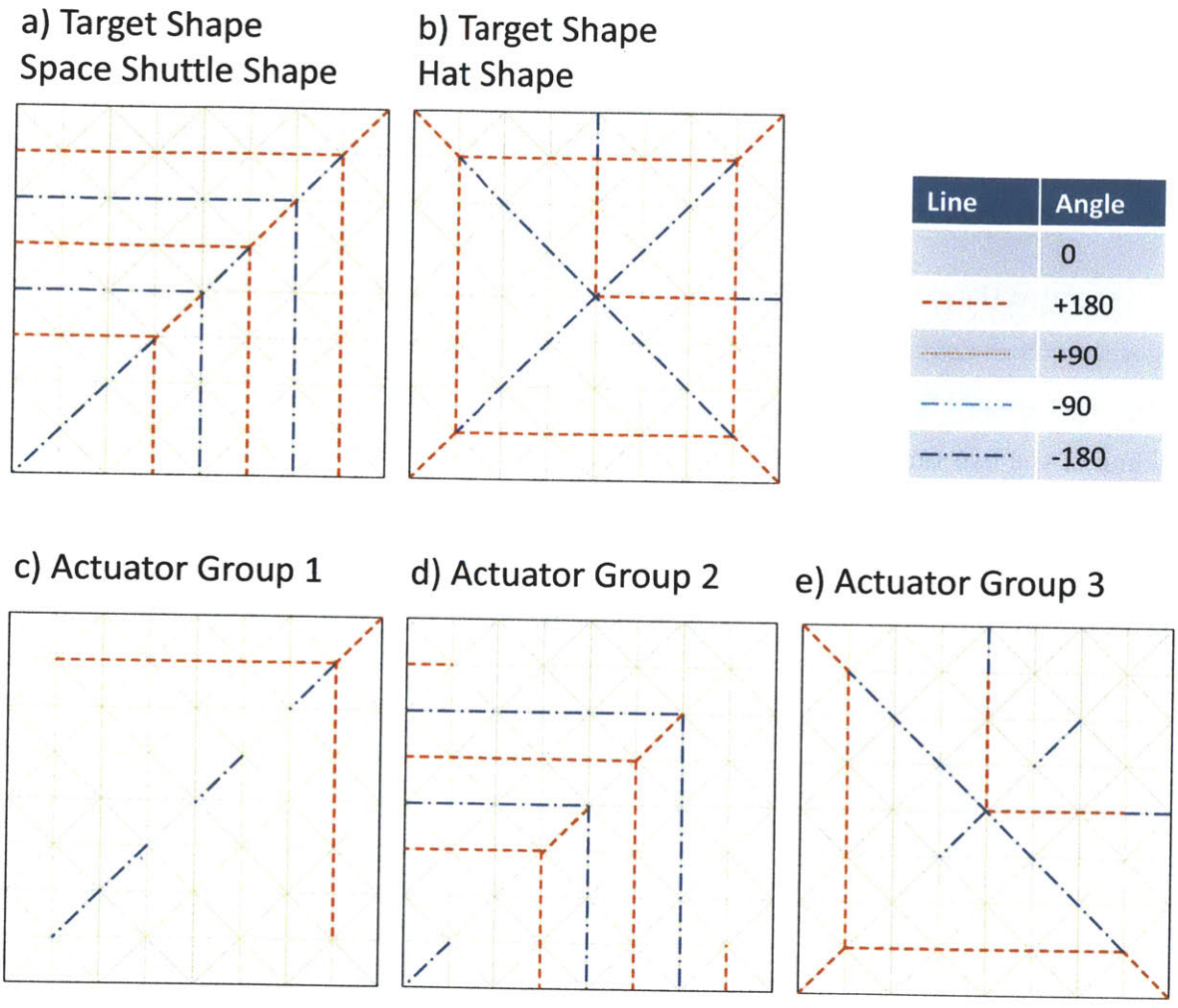


Figure 6-27: Results of origami planner (origami plan). (a)(b) are input target shapes. (c)(d)(e) are the group information of the origami plan.

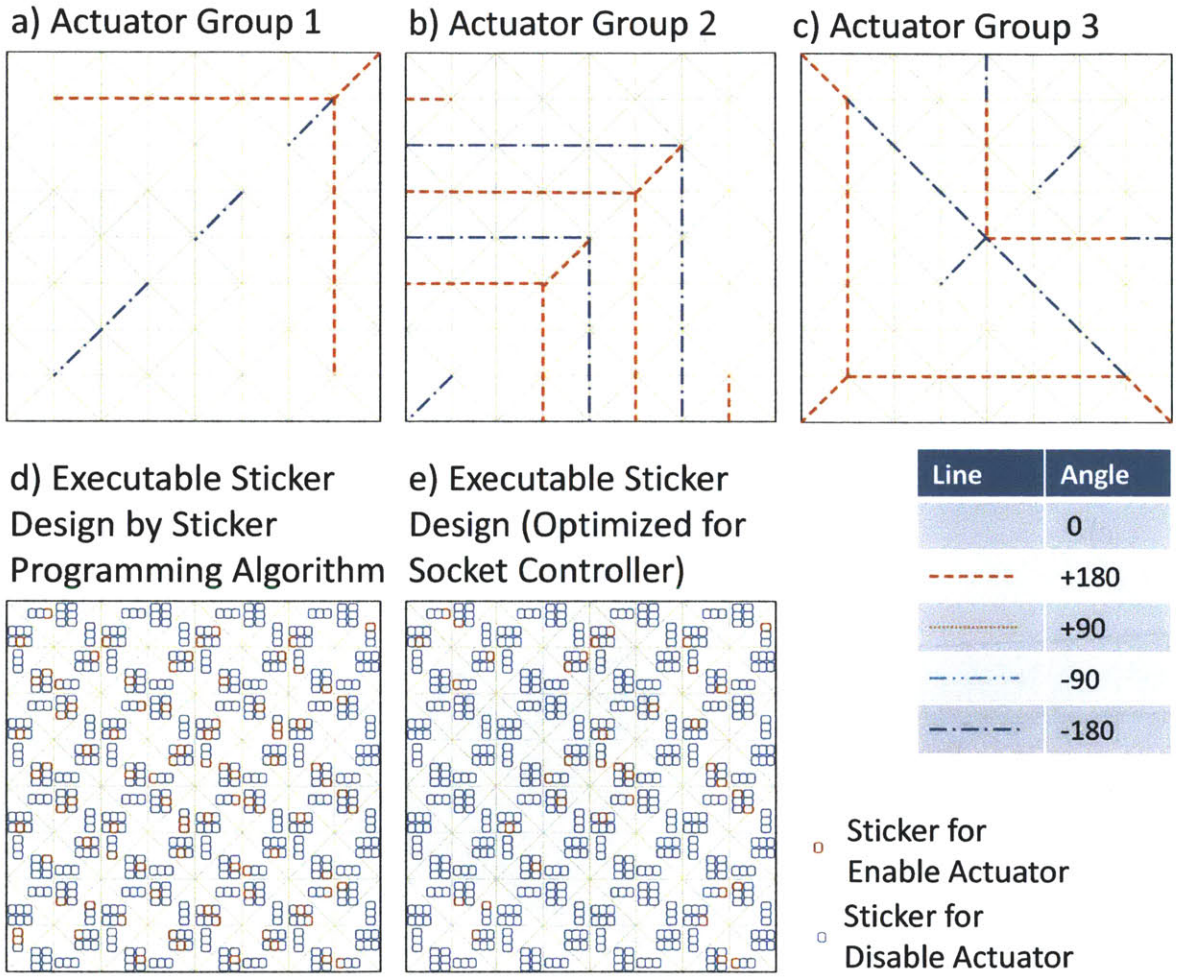


Figure 6-28: Results of Sticker Programming Algorithm (a)(b)(c) Origami Plan (d) Executable Sticker Design. (e) Executable Sticker Design intuitively optimized for 8×8 self-folding sheet. Each small square shows the sticker type for each sticker place.

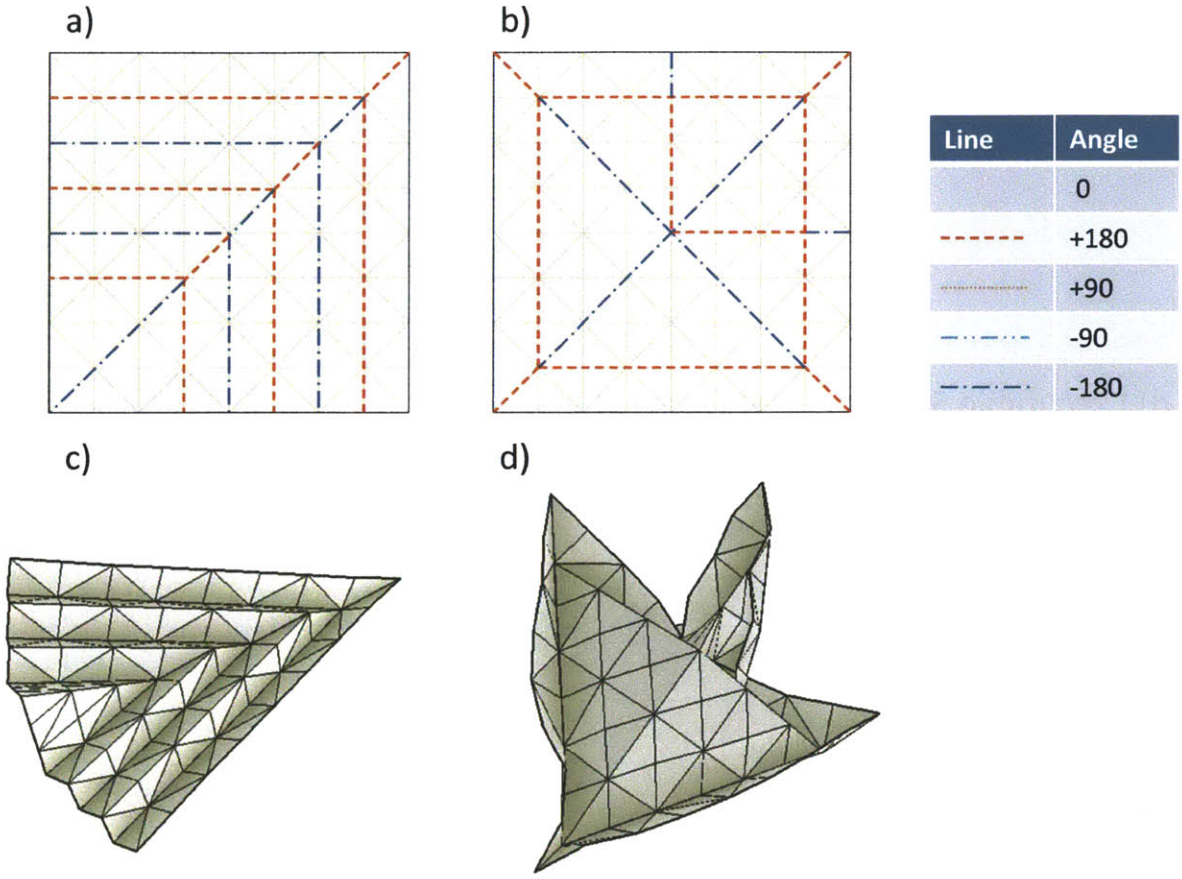


Figure 6-29: Target Shapes for Complex Motion. (a)(c) Space Shuttle (b)(d) Hat

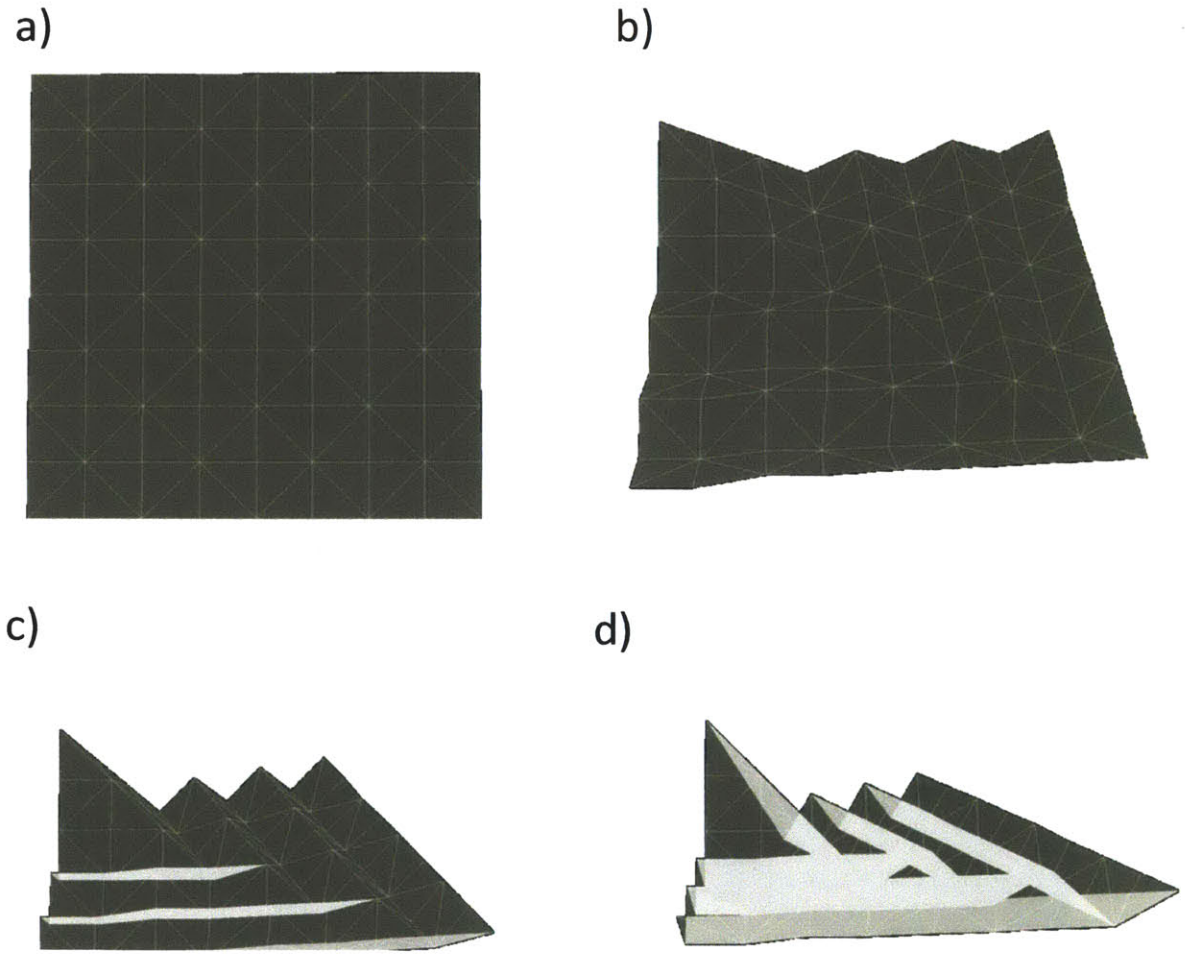


Figure 6-30: Snapshots of Space Shuttle Folding Planning [2]

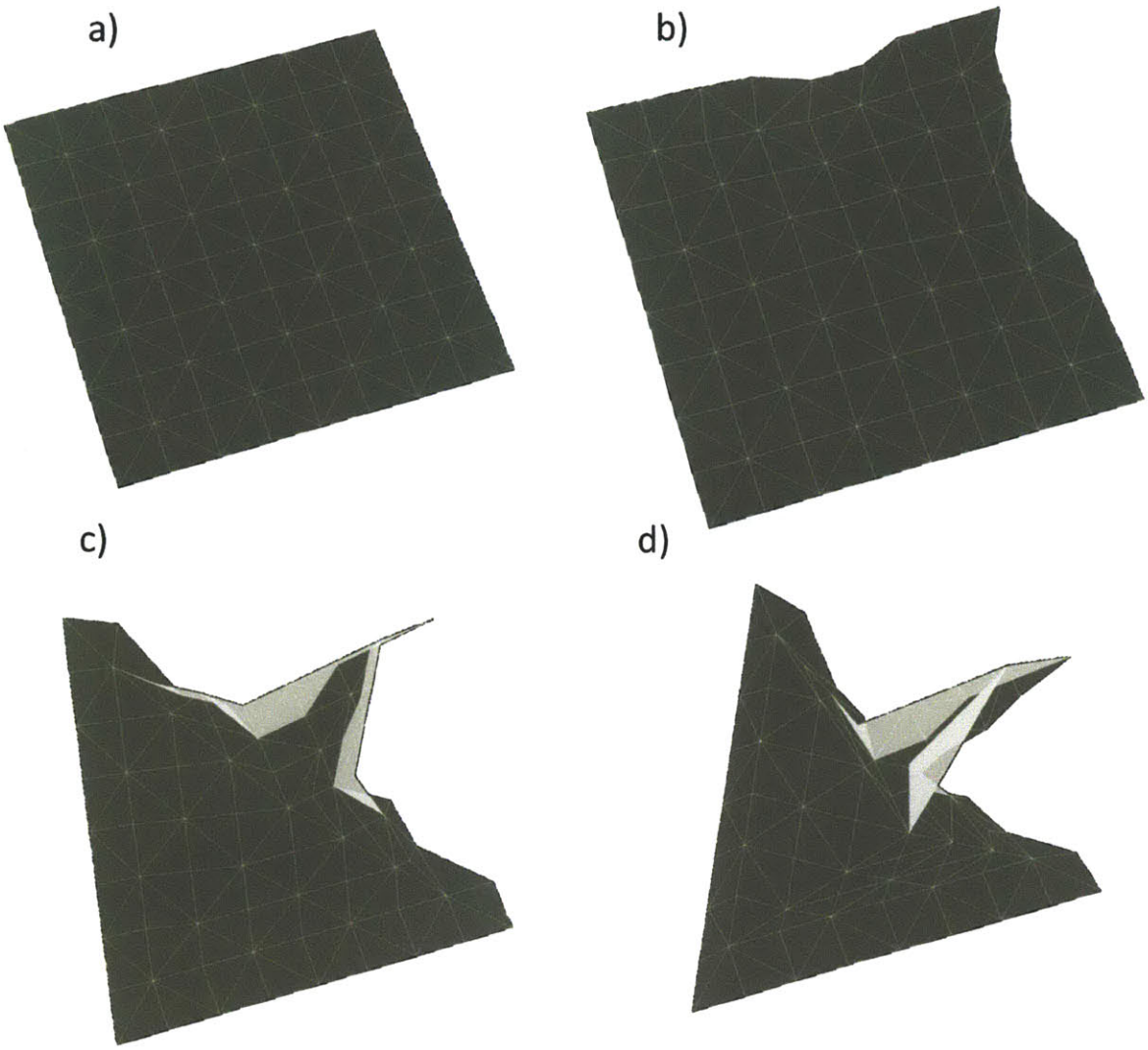


Figure 6-31: Snapshots of Hat Folding Planning. [2]

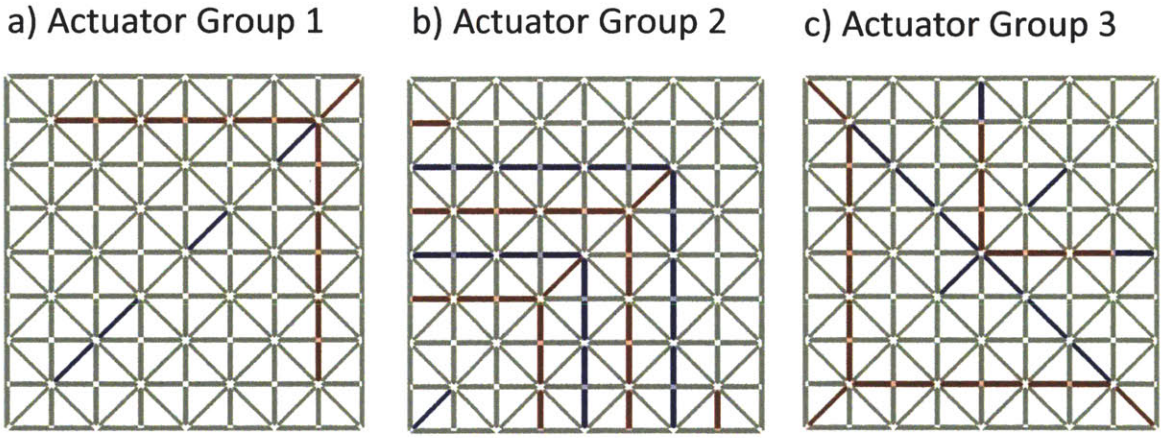


Figure 6-32: Snapshot of Sticker Program Planning. The red line denotes +180° folding. The blue line denotes -180° folding.

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

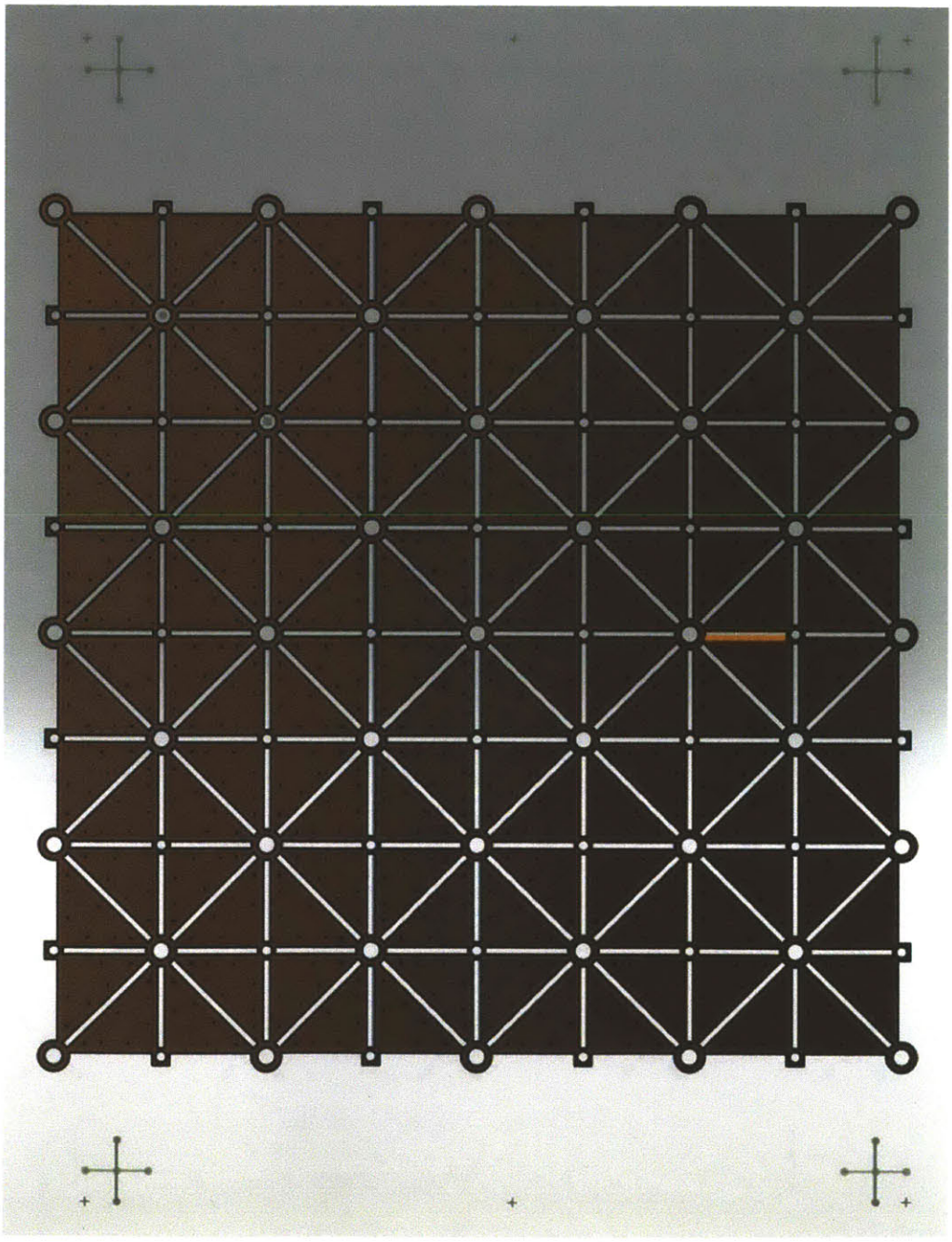
Common Parts

The Body
The Actuator

The Body

(192mm x 192mm)

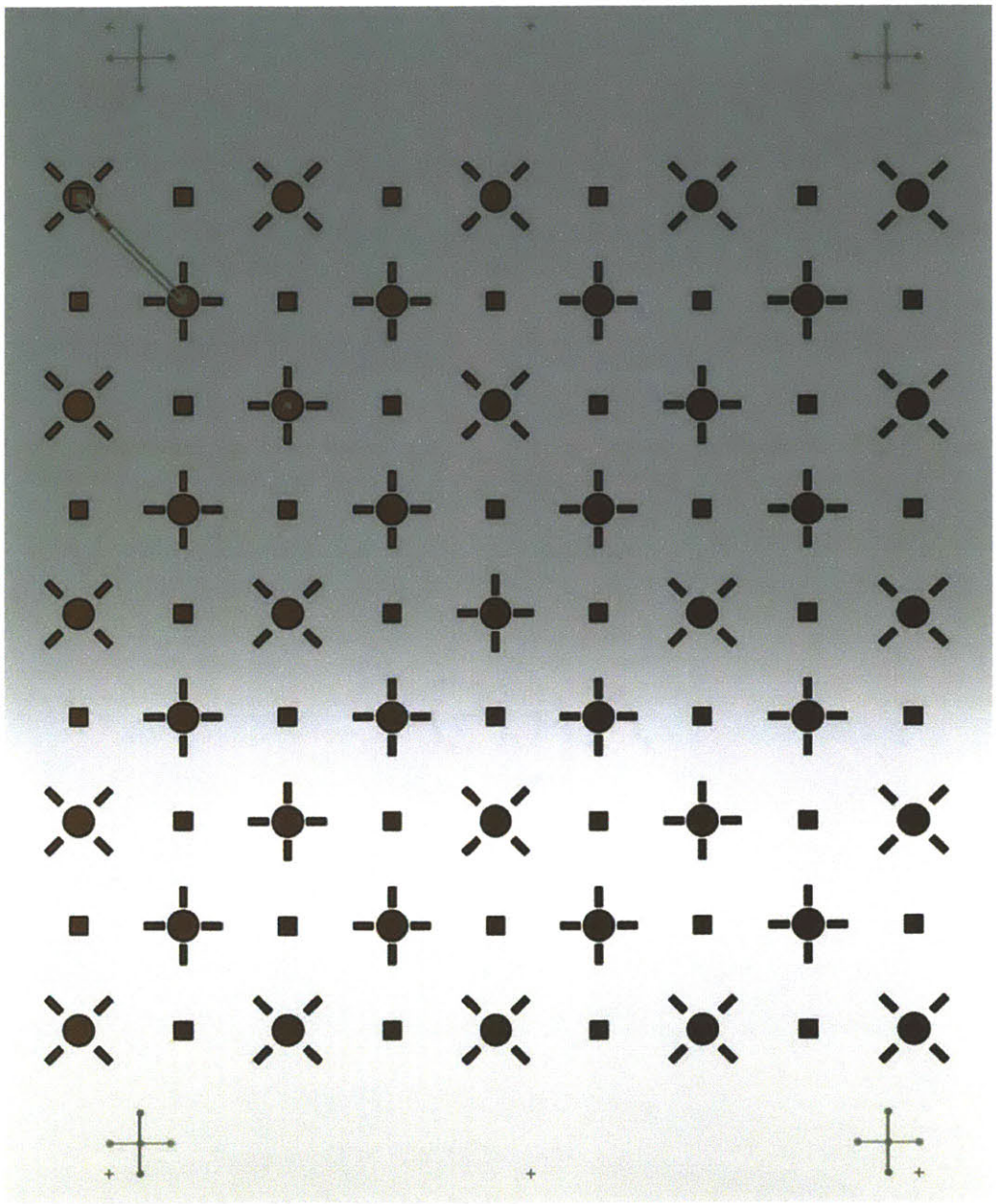
Tiles



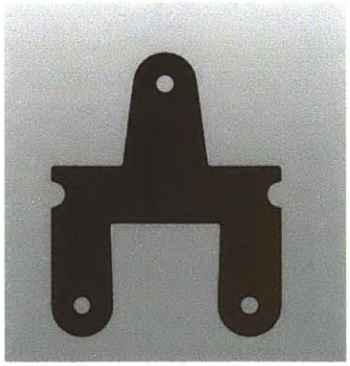
The Body

(192mm x 192mm)

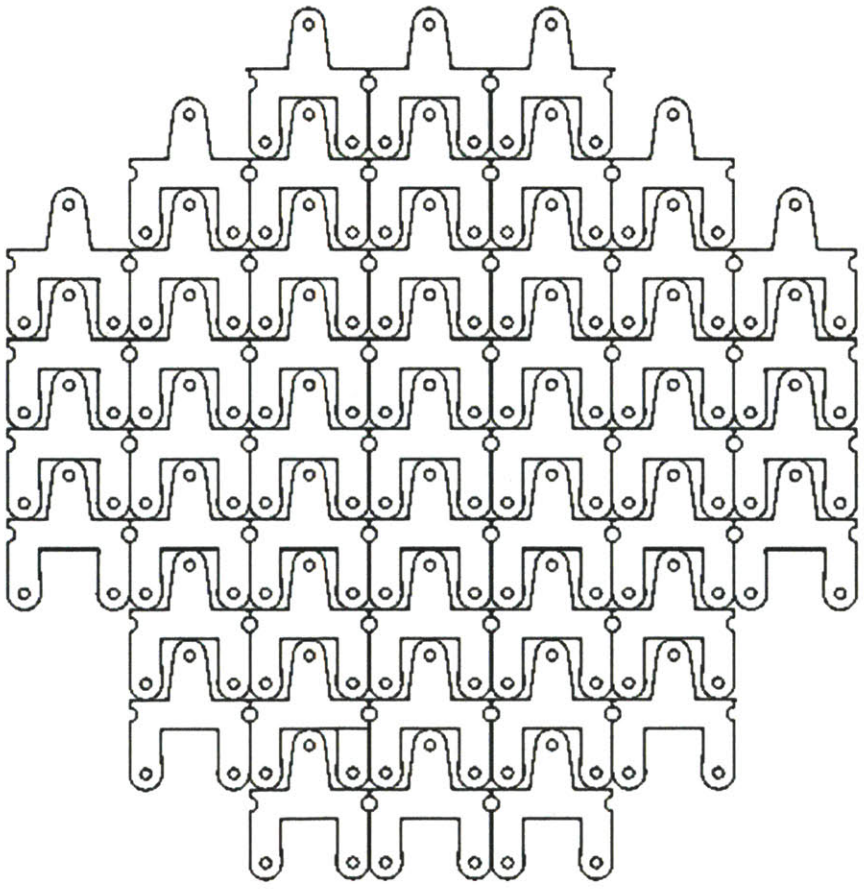
Joints



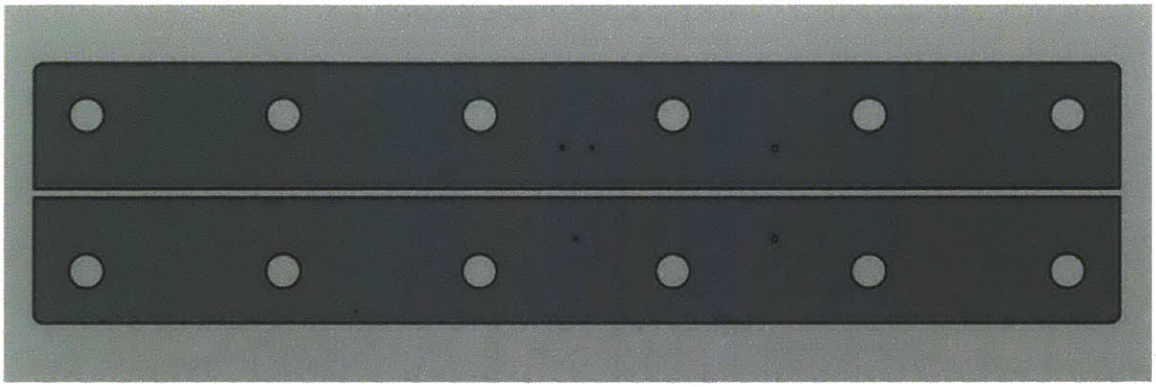
The Actuator

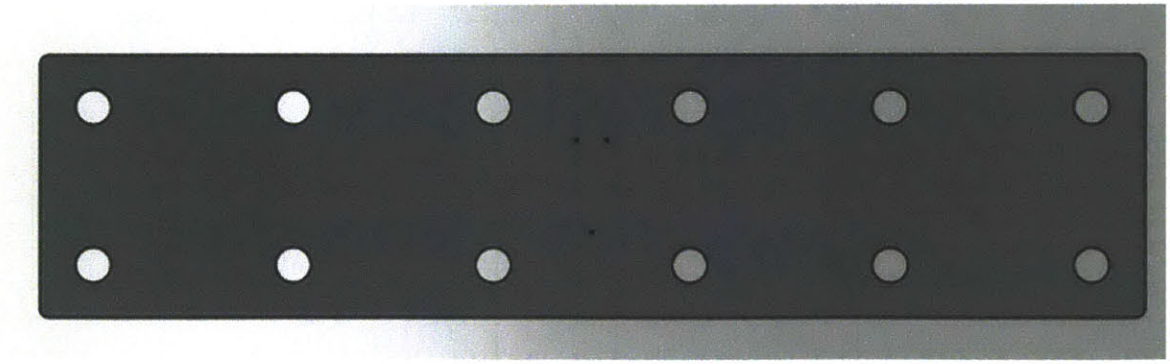


(10.23mmx8.21mm)



The Gig





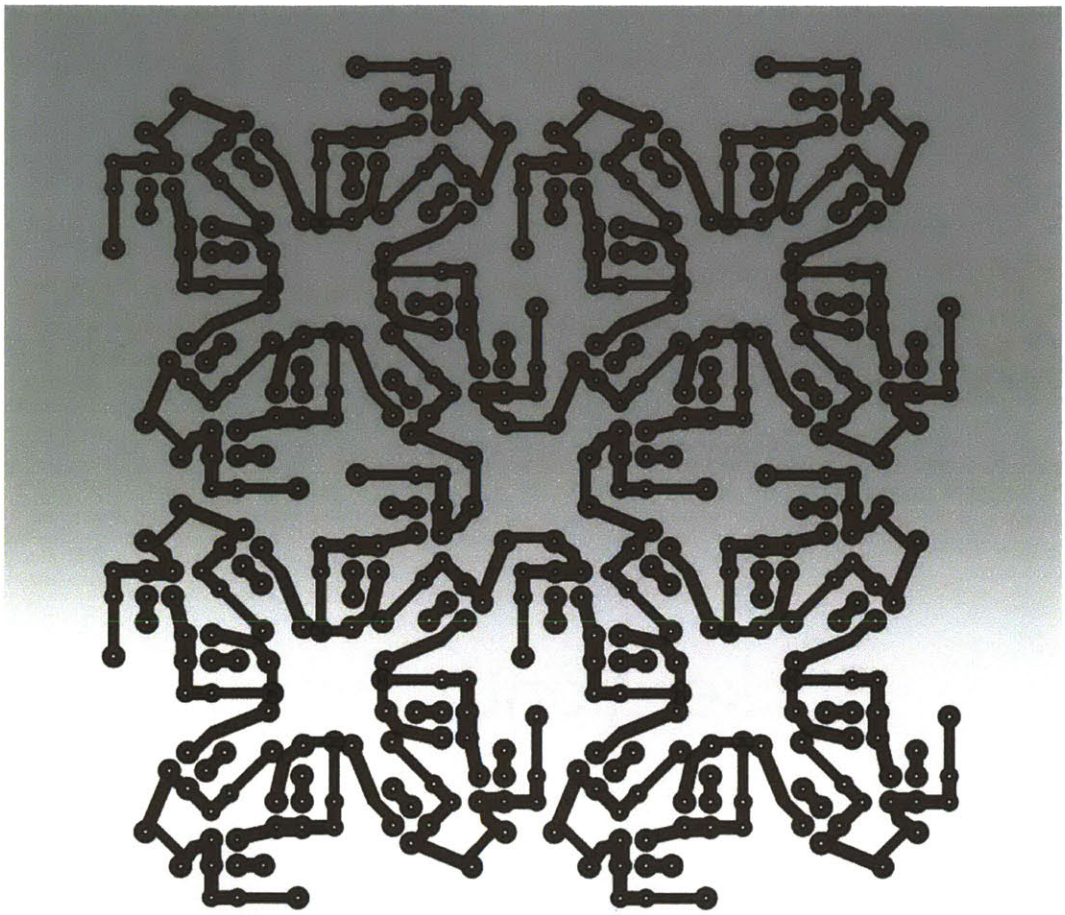
4x4 Self-Folding Sheet

(96mm x 96mm)

Sticker Controller

The Circuit

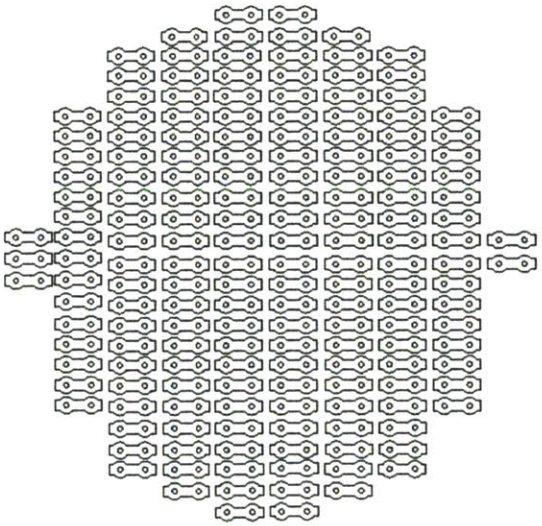
Main



Socket for Bottom

The Circuit

Patches for Executable Sticker



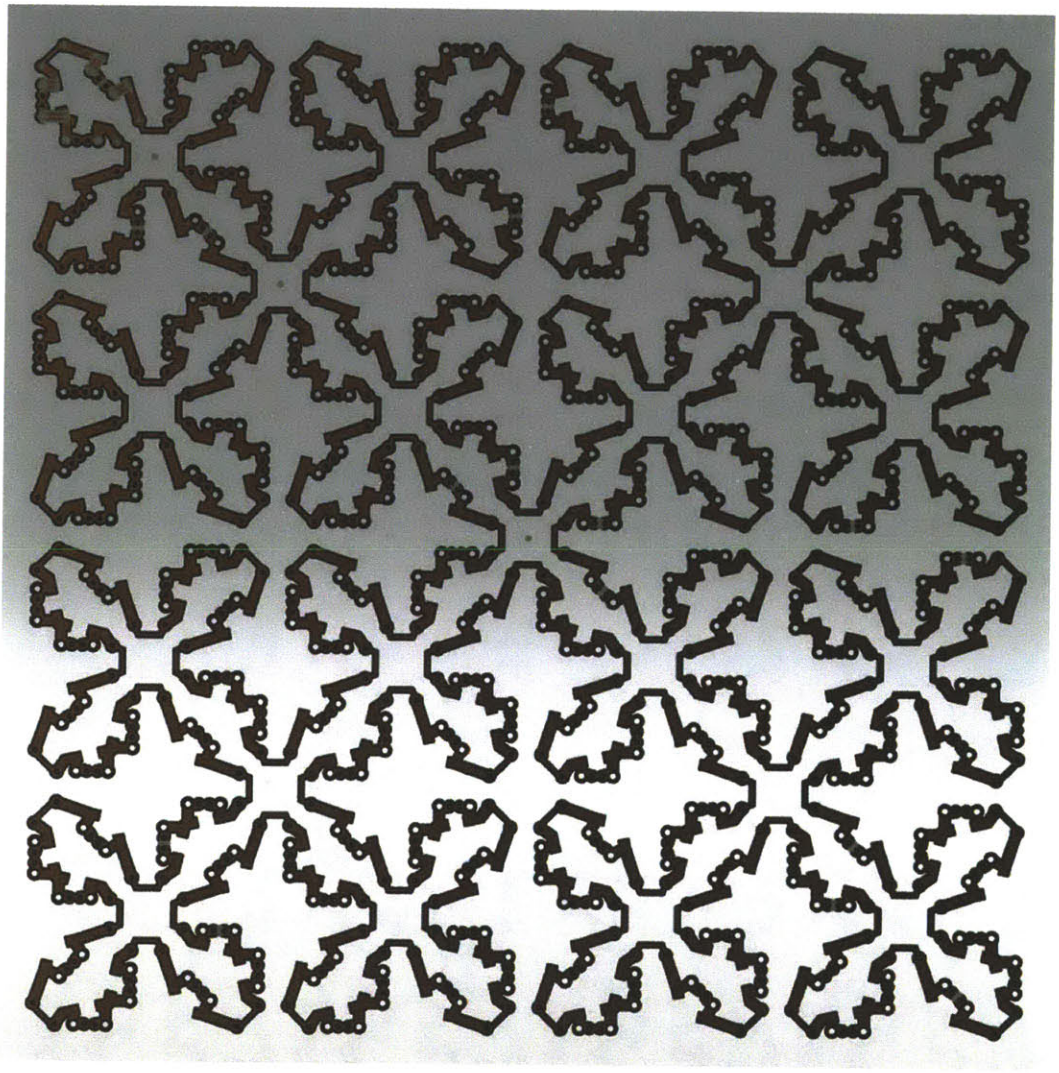
8x8 Self-Folding Sheet

(192mm x 192mm)

Sticker Controller

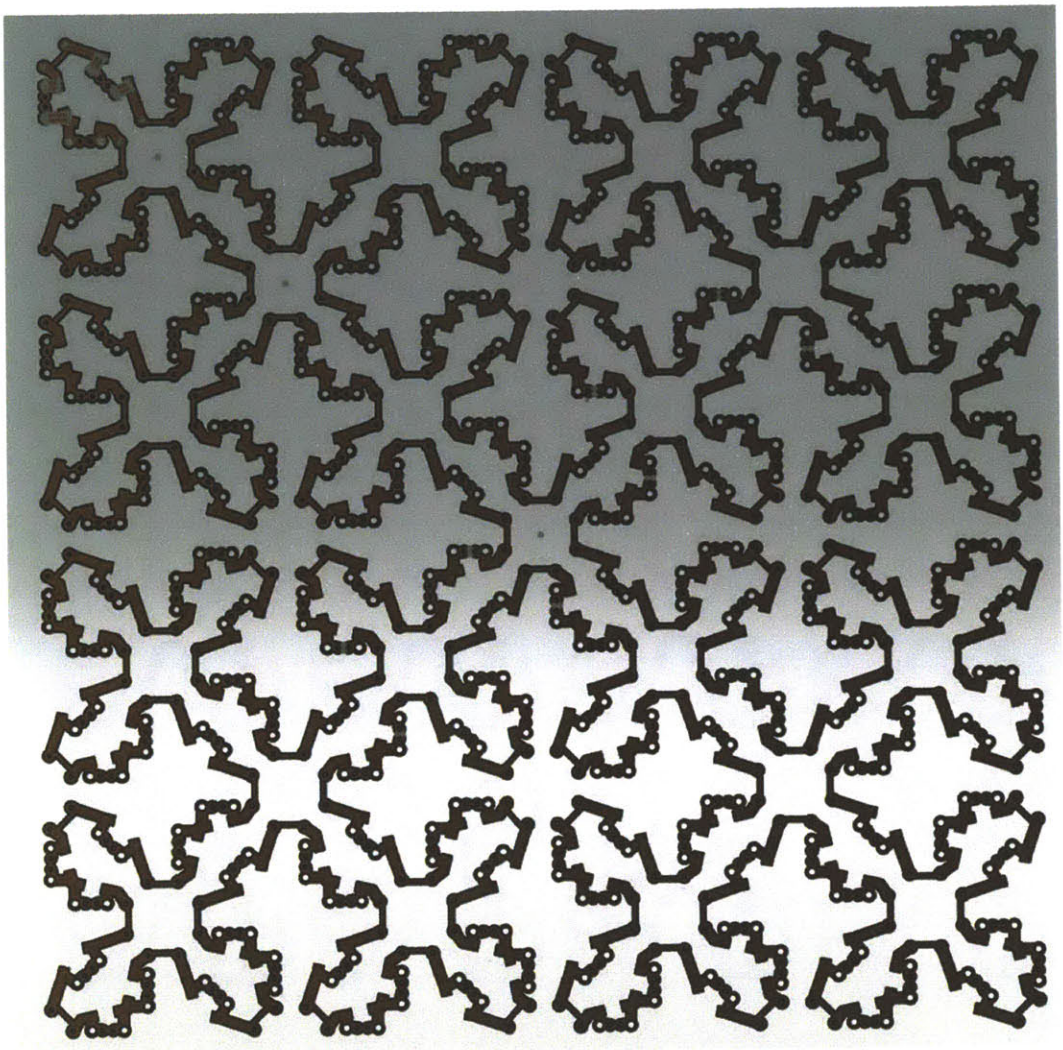
The Circuit

Layer 1



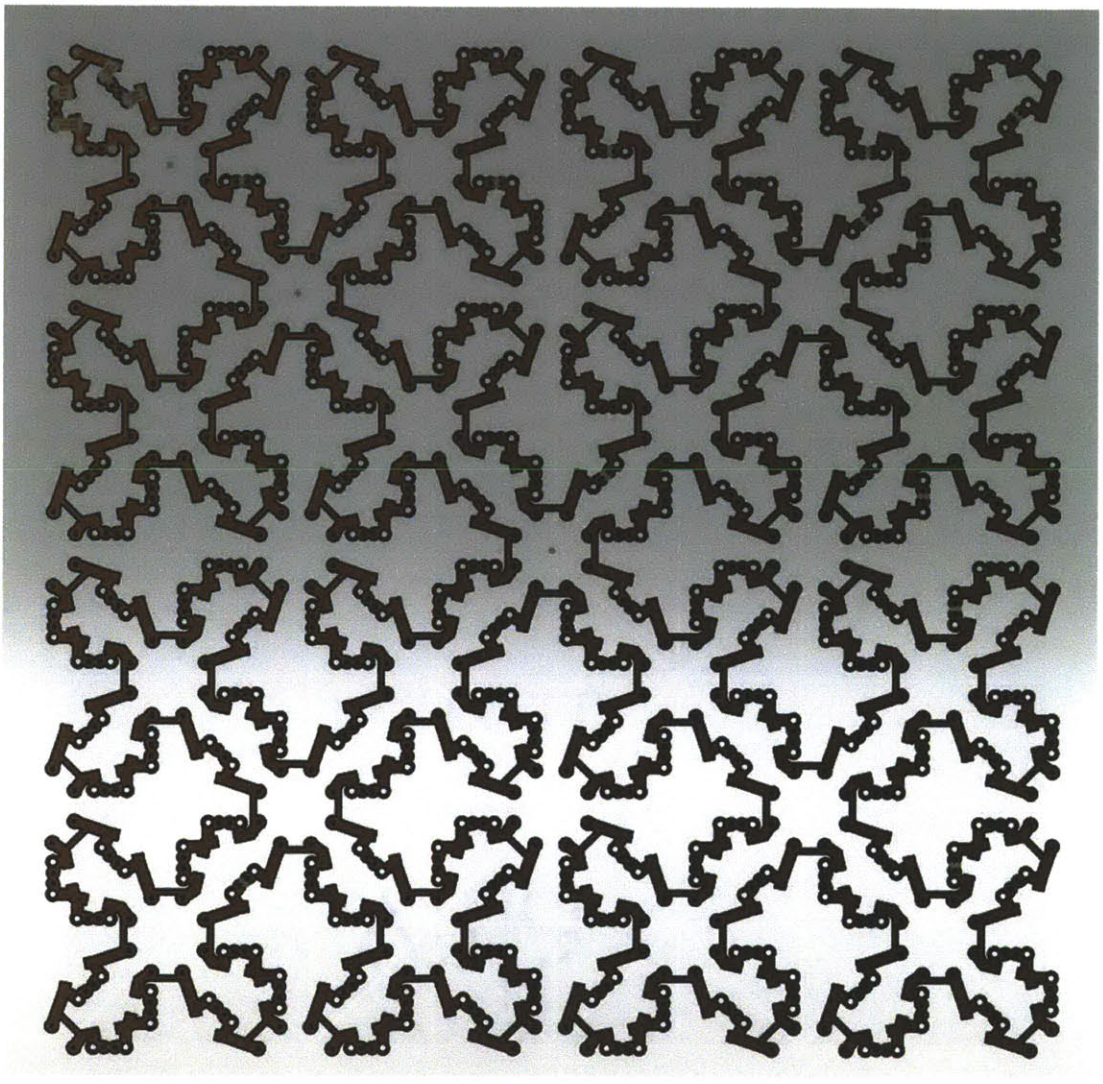
The Circuit

Layer 2

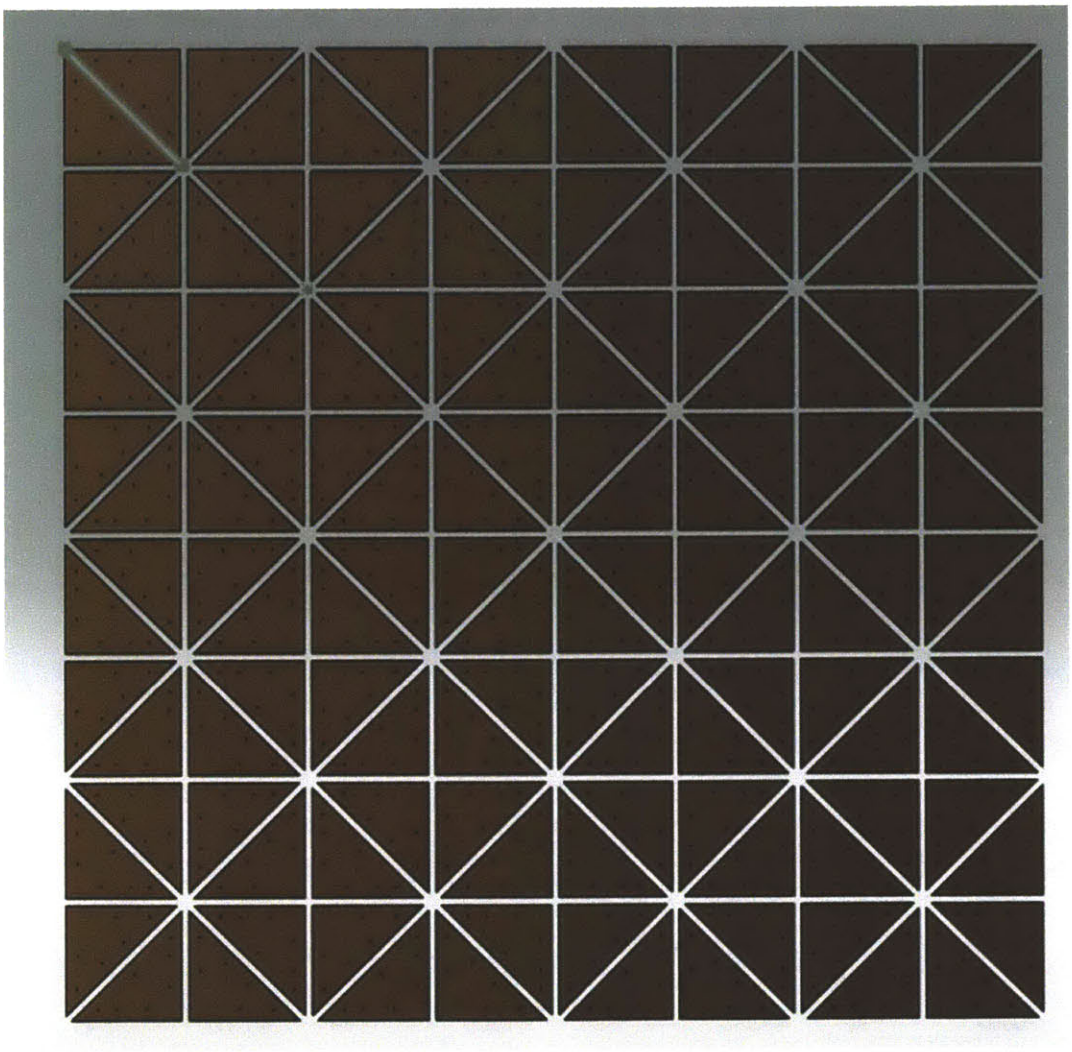


The Circuit

Layer 3



The Cover and The Insulator



Appendix C

Source Codes

```
1  package com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigami;
2
3  import sun.security.util.PendingException;
4
5  import com.drancom.programmableMatter.folding.controller.paper.Paper;
6  import com.drancom.programmableMatter.folding.dataFile.FilePlan;
7  import com.drancom.programmableMatter.folding.dataFile.FilePlanForWiring;
8  import com.drancom.programmableMatter.folding.origami.planner.Plan;
9  import com.drancom.programmableMatter.folding.origami.planner.Planner;
10
11 public class PlanerForOrigami implements Planner {
12 //   Default: percentage of level0 = 33, percentage of level1 = 66,
       percentage of level2 = 100
13 //   NoiseNumber = 0.01
14
15   final static float PERCENTAGE_OF_LEVEL0 = 0.33f;
16   final static float PERCENTAGE_OF_LEVEL1 = 0.66f;
17   final static float PERCENTAGE_OF_LEVEL2 = 1.0f;
18
19   final static float NOISE_NUMBER= 0.01f;
20
21   PlanForOrigami planForOrigami;
22
23   @Override
24   public void build(Paper[] papers) {
25
26 //     Input:  Angledata[][][]
27     float angleData[][] = new float[papers[0].getNumberOfEdges()][papers.
         length];
28 //     Output: Plan[phase][numberOfEdge], numberOfPhase
29     this.planForOrigami = new PlanForOrigami();
30     int plan[][] = new int[papers.length][papers[0].getNumberOfEdges()];
31     int numberOfPhases;
32
33     float max[] = new float [papers[0].getNumberOfEdges()];
34     float min[] = new float [papers[0].getNumberOfEdges()];
35     float standard[] = new float [papers[0].getNumberOfEdges()];
36
37
38     int angleLevel[][] = new int [papers[0].getNumberOfEdges()][papers.
         length];
39     int angleDifference[][] = new int [papers[0].getNumberOfEdges()][papers
         .length];
40
41     int phase;
42     boolean isBuildingPlan;
43     boolean isAllZero;
44
45     int i;
46     int j;
47     int k=0;
48
49     for (i=0; i < papers[0].getNumberOfEdges(); i++){
50       for (j=0; j < papers.length; j++) {
51         angleData[i][j] = papers[j].getLine(i).getAngle();
52         angleLevel[i][j] = 0;
53         if (angleData[i][j] <0) {
54           k++;
55         }
56       }
57     }
58
59 //     1.  If ABS(angledata[0..n][0..t]) < NoiseNumber), angledata[0..n][0..
       t] <- 0
60     for (i=0; i < papers[0].getNumberOfEdges(); i++){
61       for (j=0; j < papers.length; j++) {
62         if(Math.abs(angleData[i][j]) < NOISE_NUMBER) {
63           angleData[i][j] = 0.0f;
64         }
65       }
66     }
67
68
69 //     2.  Get a Max[0..n] from edges  angles.
70 //     3.  Get a Min[0..n] from edges  angles.
71     for (i=0; i< papers[0].getNumberOfEdges(); i++) {
72       max[i] = 0.0f;
73       min[i] = 0.0f;
74       standard[i] = 0.0f;
75     }
76
77     for (i=0; i < papers[0].getNumberOfEdges(); i++){
78       for (j=0; j < papers.length; j++) {
79         if(max[i] < angleData[i][j]) {
80           max[i] = angleData[i][j];
81         }
82         if(min[i] > angleData[i][j]) {
83           min[i] = angleData[i][j];
84         }
85       }
86     }
87
88
89 //     4.  If ABS(Max[0..n]) > ABS(MIN[0..n]), Standard[0..n] <- Max[0..n];
       otherwise, Standard[0..n] <- Min[0..n]
90     for (i=0; i < papers[0].getNumberOfEdges(); i++){
91       if(Math.abs(max[i]) > Math.abs(min[i])) {
92         standard[i] = max[i];
93       } else {
94         standard[i] = min[i];
95       }
96     }
97
98 //     5.  Make Anglelevel[n][t] from AngleData[n][t]:
99 //         If AngleData[i][j] = 0 or Standard[i] = 0, AngleLevel[i][j] = 0;
100 //         otherwise,  If Standard[i] > 0, AngleLevel[I][j] =  2
101 //                 If  AngleData[i][j] < Standard[i] * percentage of level1,
       AngleLevel[I][j] =  1
102 //                 If  AngleData[i][j] < Standard[i] * percentage of
       level0, AngleLevel[I][j] = 0
103 //         otherwise,  If Standard[i] < 0, AngleLevel[I][j] =  -2
104 //                 If  AngleData[i][j] > Standard[i] * percentage of level1,
       AngleLevel[I][j] =  -1
105 //                 If  AngleData[i][j] > Standard[i] * percentage of
       level0, AngleLevel[I][j] = 0
106
107    for (i=0; i < papers[0].getNumberOfEdges(); i++){
108      for (j=0; j < papers.length; j++) {
109        if (angleData[i][j] == 0.0f || standard[i] == 0.0f) {
110          angleLevel[i][j] = 0;
111        } else if (standard[i] > 0 ) {
112          angleLevel[i][j] = 2;
113          if (angleData[i][j] < standard[i] * PERCENTAGE_OF_LEVEL1) {
114            angleLevel[i][j] = 1;
115            if(angleData[i][j] < standard[i] * PERCENTAGE_OF_LEVEL0) {
116              angleLevel[i][j] = 0;
117            }
118          }
119        } else if (standard[i] < 0 ) {
120          angleLevel[i][j] = -2;
121          if (angleData[i][j] > standard[i] * PERCENTAGE_OF_LEVEL1) {
122            angleLevel[i][j] = -1;
123            if(angleData[i][j] > standard[i] * PERCENTAGE_OF_LEVEL0) {
124              angleLevel[i][j] = 0;
125            }
126          }
127        }
```

```
        }
    }

//    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]


```

```
12    int numberOfEdges;
13    int numberOfPhases;
14
15    int angleOfRotation = 0;
16
17    boolean isInvert = false;
18
19    public int[][] getPlanTable(){
20      return planTable;
21    }
22
23    public float[][] getEdgeTable() {
24      return edgeTable;
25    }
26
27    public void setPlanTable(int[][] planTable){
28      int i;
29      int j;
30
31      this.planTable = new int [planTable.length][planTable[0].length];
32
33      for (i=0; i<planTable.length ; i++) {
34        for(j=0; j< planTable[i].length ; j++) {
35          this.planTable[i][j] = planTable[i][j];
36        }
37      }
38    }
39
40    public void setEdgeTable(float[][] edgeTable){
41      int i;
42      int j;
43
44      this.edgeTable = new float[edgeTable.length][edgeTable[0].length];
45
46      for (i=0; i < edgeTable.length ; i++) {
47        for(j=0; j < edgeTable[i].length ; j++) {
48          this.edgeTable[i][j] = edgeTable[i][j];
49        }
50      }
51    }
52
53    public int getNumberOfPhases() {
54      return numberOfPhases;
55    }
56
57    public void setNumberOfPhases(int numberOfPhases) {
58      this.numberOfPhases = numberOfPhases;
59    }
60
61    public int getNumberOfEdges() {
62      return numberOfEdges;
63    }
64
65    public void setNumberOfEdges(int numberOfEdges) {
66      this.numberOfEdges = numberOfEdges;
67    }
68
69
70    public void invert() {
71      int i;
72      int j;
73
74      for (i=0; i<planTable.length ; i++) {
75        for(j=0; j< planTable[i].length ; j++) {
76          this.planTable[i][j] = planTable[i][j] * -1;
77        }
78      }
79      if (isInvert) {
80        isInvert = false;
81      } else {
82        isInvert = true;
83      }
84    }
85
86    public boolean isInvert(){
87      return isInvert;
88    }
89
90    public void rotateClockwise() {
91      int i;
92
93      for (i=0; i<edgeTable.length ; i++) {
94        this.edgeTable[i][0] = -1 * edgeTable[i][1]; // x0' = -y0
95        this.edgeTable[i][1] =      edgeTable[i][0] + 1; // y0' =  x0 + 1
96        this.edgeTable[i][2] = -1 * edgeTable[i][3]; // x1' = -y1
97        this.edgeTable[i][3] =      edgeTable[i][2] + 1; // y1' =  x1 + 1
98      }
99      angleOfRotation += 90;
100     angleOfRotation %= 360;
101
102     sort();
103   }
104
105   public void rotateCounterclockwise() {
106     int i;
107
108     for (i=0; i<edgeTable.length ; i++) {
109       this.edgeTable[i][0] =  1 * edgeTable[i][1] + 1 ; // x0' = -y0
110       this.edgeTable[i][1] = -1 * edgeTable[i][0]; // y0' =  x0 + 1
111       this.edgeTable[i][2] =  1 * edgeTable[i][3] + 1; // x1' = -y1
112       this.edgeTable[i][3] = -1 * edgeTable[i][2]; // y1' =  x1 + 1
113     }
114     angleOfRotation += 270;
115     angleOfRotation %= 360;
116
117     sort();
118   }
119
120   public void sort() {
121
122     int i;
123     int j;
124     int k;
125
126     float x0,y0;
127     float x1,y1;
128     float edgeData;
129     int   tempPlanData;
130
131     // sort startPoint and endPoint
132
133     for (i=0; i < numberOfEdges; i++){
134       x0 = edgeTable[i][0];
135       y0 = edgeTable[i][1];
136       x1 = edgeTable[i][2];
137       y1 = edgeTable[i][3];
138
139       if (x0 == x1) {
140         if (y0 < y1) {
141
142           edgeTable[i][0] = x0;
143           edgeTable[i][1] = y0;
144           edgeTable[i][2] = x1;
145           edgeTable[i][3] = y1;
146
147         } else {
148
149           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);
    }
}
```

```
1  package com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigami;
2
3  import com.drancom.programmableMatter.folding.controller.paper.Paper;
4  import com.drancom.programmableMatter.folding.dataFile.FileObj;
```

```
5  import com.drancom.programmableMatter.folding.monitor.MainWindow;
6  import com.drancom.programmableMatter.folding.monitor.
       MainWindowForFoldingRobotWiring;
7  import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
       ;
8  import com.drancom.programmableMatter.folding.origami.planner.Planner;
9  import com.drancom.programmableMatter.folding.origami.planner.
       PlannerForWiring;
10
11 public class SimulatorForOrigami {
12
13   /** /
14     public static final String PLAN_FILENAME = "c:\\foldingdata\\
           save_airplain\\plan_for_origami_airplain.csv";
15     public static final String FILENAME = "c:\\foldingdata\\save_airplain\\
           m%05d.obj";
16     public static final int NUMBER_OF_FILES= 50;
17
18   /** /
19     public static final String PLAN_FILENAME = "c:\\foldingdata\\save_box\\
           plan_for_origami_box.csv";
20     public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d
           .obj";
21     public static final int NUMBER_OF_FILES= 70;
22
23   /** /
24     public static final String PLAN_FILENAME = "c:\\foldingdata\\
           save_sailboat2\\plan_for_origami_sailboat2.csv";
25     public static final String FILENAME = "c:\\foldingdata\\save_sailboat2
           \\m%05d.obj";
26     public static final int NUMBER_OF_FILES= 35;
27
28   /**/
29     public static final String PLAN_FILENAME = "c:\\foldingdata\\save_bench
           \\plan_for_origami_save_bench.csv";
30     public static final String FILENAME = "c:\\foldingdata\\save_bench\\m
           %05d.obj";
31     public static final int NUMBER_OF_FILES= 70;
32
33   /**/
34
35 //     public static final String FILENAME = "c:\\foldingdata\\save_box\\
       m00070.obj";
36 //     public static final int NUMBER_OF_FILES= 1;
37
38     Paper[] papers;
39     FileObj[] fileObjs;
40     MainWindowForFoldingRobotOrigami mainWindow;
41     public SimulatorForOrigami() {
42
43     }
44
45     void run() {
46       int i;
47       String fileName;
48
49       // Initiation
50       papers = new Paper[NUMBER_OF_FILES];
51       fileObjs = new FileObj[NUMBER_OF_FILES];
52       mainWindow = new MainWindowForFoldingRobotOrigami();
53
54       // load
55       for (i=0; i < NUMBER_OF_FILES; i++) {
56         fileName = String.format(FILENAME, i+1);
57         papers[i] = new Paper();
58         fileObjs[i] = new FileObj();
59         fileObjs[i].load(fileName, papers[i]);
60       }
61
62       Planner planer = new PlanerForOrigami(); // new planer
63
64       planer.build(papers);
65
66       planer.exportPlan(PLAN_FILENAME, papers);
67
68       mainWindow.run(papers, (PlanForOrigami) planer.getPlan());
69
70     }
71
72     public static void main(String[] args) {
73       // TODO Auto-generated method stub
74       SimulatorForOrigami simulator = new SimulatorForOrigami ();
75       simulator.run();
76     }
77 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigami;
2
3  import java.awt.List;
4  import java.io.BufferedInputStream;
5  import java.io.BufferedReader;
6  import java.io.BufferedWriter;
7  import java.io.DataInputStream;
8  import java.io.File;
9  import java.io.FileInputStream;
10 import java.io.FileNotFoundException;
11 import java.io.FileOutputStream;
12 import java.io.FileReader;
13 import java.io.FileWriter;
14 import java.io.IOException;
15 import java.lang.reflect.Array;
16 import java.util.ArrayList;
17 import java.util.StringTokenizer;
18
19 import com.drancom.programmableMatter.folding.controller.paper.Paper;
20 import com.drancom.programmableMatter.folding.dataFile.FilePlan;
21 import com.drancom.programmableMatter.folding.origami.planner.Plan;
22 import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
       ;
23
24 public class FilePlanForOrigami implements FilePlan {
25
26   @Override
27   public void build(String fileName, Plan plan) {
28     int i;
29     int j;
30     int k;
31
32     int numberOfEdges;
33     int numberOfPhases;
34
35     File file = new File(fileName);
36     PlanForOrigami planForOrigami = (PlanForOrigami) plan;
37     int [][] planTable = planForOrigami.getPlanTable();
38     float [][] edgeTable = planForOrigami.getEdgeTable();
39
40     numberOfEdges = planForOrigami.getNumberOfEdges();
41     numberOfPhases = planForOrigami.getNumberOfPhases();
42
43     try {
44       boolean success = file.createNewFile();
45       if (success) {
46         // File did not exist and was created
47       } else {
48       }
49     } catch (IOException e) {
50       // TODO Auto-generated catch block
```

```
51          e.printStackTrace();
52        }
53
54        String bufferLine  = new String();
55
56        try {
57          BufferedWriter bufferedWriter = new BufferedWriter (new FileWriter (
                file));
58
59          // print edges
60          bufferLine = String.format("# %d edges \n", numberOfEdges); // "# %d
                phases\n"
61          bufferedWriter.write(bufferLine);
62          bufferedWriter.newLine();
63
64          bufferLine = String.format("# e startPointX startPointY endPointX
                endPointX \n", numberOfPhases); // "# %d phases\n"
65
66          // print out to file
67          bufferedWriter.write(bufferLine);
68          bufferedWriter.newLine();
69
70        for (i = 0; i < numberOfEdges; i++) {
71          bufferLine = String.format("e");
72          for (j=0 ; j < 4 ; j++) {
73            bufferLine += String.format(", %f", edgeTable[i][j]);
74          }
75
76          // print out to file
77
78          bufferedWriter.write(bufferLine);
79          bufferedWriter.newLine();
80
81        }
82
83        // print phase
84        bufferLine = String.format("# %d phases", numberOfPhases); // "# %d
                phases\n"
85        bufferedWriter.write(bufferLine);
86        bufferedWriter.newLine();
87
88        bufferLine = String.format("# p phases planData", numberOfPhases); //
                "# %d phases\n"
89
90        // print out to file
91        bufferedWriter.write(bufferLine);
92        bufferedWriter.newLine();
93
94        // planTable [phases][edgeNumber]
95        for (i = 0; i < numberOfPhases; i++) {
96          for (j = 0; j < numberOfEdges; j++) {
97            bufferLine = String.format("p");
98            bufferLine += String.format(", %d, %d", i, planTable[i][j]);
99
100           bufferedWriter.write(bufferLine);
101           bufferedWriter.newLine();
102         }
103         // print to file
104       }
105
106         // file close
107         bufferedWriter.close();
108
109       } catch (FileNotFoundException e) {
110         e.printStackTrace();
111       } catch (IOException e) {
112         e.printStackTrace();
113       }
114     }
```

```
115
116   @Override
117   public void build(String fileName, Paper[] papers) {
118   }
119
120   @SuppressWarnings("deprecation")
121   @Override
122   public void read(String fileName, Plan plan) {
123     PlanForOrigami planForOrigami = (PlanForOrigami) plan;
124
125     int i;
126     int j;
127
128     String data;
129
130     int [][] planTable;
131     float [][] edgeTable;
132
133     int numberOfPhases;
134     int numberOfEdges;
135
136     // Temporally  variable
137     ArrayList<int[]> planArrayList = new ArrayList<int[]>();
138     ArrayList<float[]> edgeArrayList = new ArrayList<float[]>();
139
140     String head;
141     float[] tempEdge;
142     int [] tempPlanData;
143     int tempPhase;
144
145     File file = new File(fileName);
146
147     numberOfPhases = 0;
148     numberOfEdges  = 0;
149
150     try {
151
152       BufferedReader bufferedReader = new BufferedReader(new FileReader(
              file));
153       // read buffer
154
155       while ((data = bufferedReader.readLine()) != null) {
156
157         StringTokenizer st = new StringTokenizer(data, ",");
158         if (st.hasMoreElements()) {
159           head =  st.nextToken();
160           if(head.startsWith("#")) {
161
162           } else if(head.equals("e")) {
163             tempEdge = new float[4];
164             tempEdge[0] = Float.parseFloat(st.nextToken());
165             tempEdge[1] = Float.parseFloat(st.nextToken());
166             tempEdge[2] = Float.parseFloat(st.nextToken());
167             tempEdge[3] = Float.parseFloat(st.nextToken());
168             edgeArrayList.add(tempEdge);
169
170           } else if(head.equals("p")) {
171             tempPlanData = new int[2];
172             tempPlanData[0] = Integer.parseInt(st.nextToken().trim());
173             tempPlanData[1] = Integer.parseInt(st.nextToken().trim());
174             planArrayList.add(tempPlanData);
175             numberOfPhases++;;
176           }
177         }
178       }
179
180       bufferedReader.close();
181
182     } 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
    }

}
```

```
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();
```

```
65          plansForOrigami[i][j].load(PLAN_FILENAME[i]);
66        }
67      }
68
69      // Optimal Algorithms
70      // input : PlansOfOrigami
71      // output : planOfOrigamisOfMinimalString;
72      //        planOfOrigamisOfMinimalThreading
73
74      // 1. for i=0 to NumberOfPlans
75      // 2.       for j=0 to 8
76      // 3.             plansOfOrigami[i][j] <- plansOfOrigami[i]
77      //
78      // 4. for i=0 to NumberOfPlans
79      // 5.       for j=0 to 8
80      // 6.             if j>=4,
81      // 7.           do invert plansOfOrigami[i][j]
82      // 8.             for k = 0 to k < j %4
83      // 9.             do rotate clockwise plansOfOrigami[i][j]
84
85      for (i=0; i < NUMBER_OF_PLAN_FILES; i++) {
86        for ( j=0; j < 8 ; j ++) {
87          if (j >= 4 ) {
88            plansForOrigami[i][j].invert();
89          }
90          for (k = 0; k < j % 4; k++) {
91            plansForOrigami[i][j].rotateClockwise();
92          }
93        }
94      }
95
96      // 10. numberOfPlansForOrigais <- 1
97      // 11. for i=0 to numberOfPlan
98      // 12       numberOfPlansForOrigamis *= 8
99      numberOfPlansForOrigamis = 1;
100     for (i = 0; i < NUMBER_OF_PLAN_FILES; i++) {
101       numberOfPlansForOrigamis *= 8;
102     }
103
104     // 0, 0 invert , 90, 90 invert 180, 180 invert , 270 invert ,  270
105     planers = new Planner[numberOfPlansForOrigamis ];
106     inputPlansForOrigami = new PlanForOrigami[NUMBER_OF_PLAN_FILES];
107     int tempInt;
108
109     // 13. for i = 0 to numberOfPlansForOrigamis
110     // 14.       tempInt <- i
111     // 15.       for j = 0 to numberOfPlan
112     // 16.         inputPlansForOrigami[j] = plansForOrigami[j][tempInt % 8]
113     // 17.         tempInt/=8;
114     // 18.       planers[i] planAlgoritmForOrigamis(inputPlansForOrigami)
115
116     for (i=0; i < numberOfPlansForOrigamis; i++ ) {
117       planers[i] = new PlanerForOrigamis(); // new planer
118
119       tempInt = i;
120       for (j = 0; j < NUMBER_OF_PLAN_FILES; j++) {;
121         inputPlansForOrigami[j] = plansForOrigami[j][tempInt % 8];
122         tempInt /= 8;
123       }
124
125       planers[i].build(inputPlansForOrigami);
126
127     }
128     // 19. optimalPlanForOrigamis <- 0
129     // 20. numberOfActiveEdgeOfOptimalPlanForOrigamis <-
              getNumberOfActiveEdgesInPlanGroup(planers[0])
130     optimalPlanForOrigamis = 0;
131     numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis) planers
            [0].getPlan()).getNumberOfActiveEdgesInPlanGroup();
132
133     // 21. optimalGroupsForOrigamis <- 0
134     // 22. numberOfGroupsOfOptimalNumberOfGroups <- get
            numberOfGroupsOfOptimalNumberOfGroups(planers[0])
135     optimalGroupsForOrigamis = 0;
136     numberOfGroupsOfOptimalNumberOfGroups = ((PlanForOrigamis) planers[0].
          getPlan()).getNumberOfActivePlanGroup();
137
138     // 23. optimalPlanForOrigamis <- number of index planers having the
            smallest number of Active Edges In PlanGroup
139     for (i = 0; i< numberOfPlansForOrigamis; i++) {
140       if (       numberOfActiveEdgeOfOptimalPlanForOrigamis >  ((
              PlanForOrigamis) planers[i].getPlan()).
              getNumberOfActiveEdgesInPlanGroup()
141           ||  ((numberOfActiveEdgeOfOptimalPlanForOrigamis == ((
                 PlanForOrigamis) planers[i].getPlan()).
                 getNumberOfActiveEdgesInPlanGroup())
142               && (((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan
                     ()).getNumberOfActivePlanGroup()
143               >   ((PlanForOrigamis) planers[i              ].getPlan()).
                     getNumberOfActivePlanGroup())))) {
144         optimalPlanForOrigamis = i;
145         numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis)
              planers[i].getPlan()).getNumberOfActiveEdgesInPlanGroup();
146       }
147
148     // 24. optimalGroupsForOrigamis <- number of index of planers having
            the smallest number of Active PlanGroup
149       if (       numberOfGroupsOfOptimalNumberOfGroups >  ((PlanForOrigamis)
              planers[i].getPlan()).getNumberOfActivePlanGroup()
150           ||  ((numberOfGroupsOfOptimalNumberOfGroups == ((PlanForOrigamis)
                 planers[i].getPlan()).getNumberOfActivePlanGroup())
151               && (((PlanForOrigamis) planers[optimalGroupsForOrigamis].
                     getPlan()).getNumberOfActiveEdgesInPlanGroup()
152               >   ((PlanForOrigamis) planers[i                ].getPlan()).
                     getNumberOfActiveEdgesInPlanGroup())))) {
153         optimalGroupsForOrigamis = i;
154         numberOfGroupsOfOptimalNumberOfGroups = ((PlanForOrigamis) planers[
              i].getPlan()).getNumberOfActivePlanGroup();
155       }
156
157       System.out.printf("planForOrigamis \t %d %o %d %d %f %f %d %d ", i, i
158           , ((PlanForOrigamis) planers[i].getPlan()).
                 getNumberOfActiveEdgesInPlanGroup()
159           , ((PlanForOrigamis) planers[i].getPlan()).
                 getNumberOfActiveEdgesInPlansForOrigami()
160           , (float)((PlanForOrigamis) planers[i].getPlan()).
                 getNumberOfActiveEdgesInPlanGroup() / (float) ( (
                 PlanForOrigamis) planers[i].getPlan()).
                 getNumberOfActiveEdgesInPlansForOrigami()
161           , (float)((PlanForOrigamis) planers[i].getPlan()).
                 getNumberOfActiveEdgesInPlanGroup() / (float)( ((
                 PlanForOrigamis) planers[i].getPlan()).getNumberOfEdges() *
                 NUMBER_OF_PLAN_FILES)
162           , ((PlanForOrigamis) planers[i].getPlan()).
                 getNumberOfActivePlanGroup()
163           , ((PlanForOrigamis) planers[i].getPlan()).getNumberOfGroups());
164
165       if(((PlanForOrigamis) planers[i].getPlan()).isAlined()){
166         System.out.printf("true\n");
167         } else {
168           System.out.printf("false\n");
169         }
170     }
171
172     System.out.printf("optimalEdgePlanForOrigamis \t %d %o %d %d %f %f %d %
            d\n", optimalPlanForOrigamis, optimalPlanForOrigamis
173           , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                   .getNumberOfActiveEdgesInPlanGroup()
```

```
174             , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                    .getNumberOfActiveEdgesInPlansForOrigami()
175             , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
                    getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)((
                    PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan()
                    ).getNumberOfActiveEdgesInPlansForOrigami()
176             , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
                    getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)
                    (((PlanForOrigamis) planers[optimalPlanForOrigamis].
                    getPlan()).getNumberOfEdges() * NUMBER_OF_PLAN_FILES)
177             , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                    .getNumberOfActivePlanGroup()
178             , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                    .getNumberOfGroups());
179
180     System.out.printf("optimalNumberOfGroups \t %d %o %d %d %f %f %d %d\n",
            optimalGroupsForOrigamis, optimalGroupsForOrigamis
181         , ((PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
                getNumberOfActiveEdgesInPlanGroup()
182         , ((PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
                getNumberOfActiveEdgesInPlansForOrigami()
183         , (float)((PlanForOrigamis) planers[optimalGroupsForOrigamis].
                getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)((
                PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
                getNumberOfActiveEdgesInPlansForOrigami()
184         , (float)((PlanForOrigamis) planers[optimalGroupsForOrigamis].
                getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)(((
                PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
                getNumberOfEdges() * NUMBER_OF_PLAN_FILES)
185         , ((PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
                getNumberOfActivePlanGroup()
186         , ((PlanForOrigamis) planers[optimalGroupsForOrigamis].getPlan()).
                getNumberOfGroups());
187
188     planers[optimalPlanForOrigamis].exportPlan(String.format(
            PLAN_FOR_ORIGAMIS_FILENAME, "optimalEdgePlan_",
            NUMBER_OF_PLAN_FILES, PLAN_FOR_ORIGAMIS_FILETYPE));
189
190     planers[optimalGroupsForOrigamis].exportPlan(String.format(
            PLAN_FOR_ORIGAMIS_FILENAME, "optimalNumberOfActiveGroups_",
            NUMBER_OF_PLAN_FILES,PLAN_FOR_ORIGAMIS_FILETYPE));
191
192 //     monitor.run((PlanForOrigamis) planers[optimalGroupsForOrigamis].
        getPlan());
193 //     monitor.run((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan
        ());
194 //     monitor.run((PlanForOrigamis) planers[8].getPlan());
195 //     monitor.run((PlanForOrigamis) planers[19].getPlan());
196
197   }
198
199   public static void main(String[] args) {
200     SimulatorToFindOptimalOrigamisWithInvertingAndRotation simulator = new
            SimulatorToFindOptimalOrigamisWithInvertingAndRotation ();
201     simulator.run();
202   }
203
204 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamis;
2
3  import com.drancom.programmableMatter.folding.dataFile.FileObj;
4  import com.drancom.programmableMatter.folding.monitor.
       MonitorOfPlanGroupOfPlanForOrigamis;
5  import com.drancom.programmableMatter.folding.origami.planner.Plan;
6  import com.drancom.programmableMatter.folding.origami.planner.Planner;
7  import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
       .PlanForOrigami;
8
9
10 public class SimulatorToFindOptimalOrigamisWithInverting {
11
12   public static final String PLAN_FILENAME[] = {
13       "c:\\foldingdata\\save_8x8table\\plan_for_origami_save_8x8table.csv"
14     , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"
15     , "c:\\foldingdata\\save_8x8airplain\\plan_for_origami_8x8airplain.csv"
16     , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"
17     , "c:\\foldingdata\\save_8x8sailboat\\plan_for_origami_8x8sailboat.csv"
18     , "c:\\foldingdata\\save_8x8bench\\plan_for_origami_save_8x8bench.csv"
         };
19
20   public static final int NUMBER_OF_PLAN_FILES= 2;
21
22   public static final String PLAN_FOR_ORIGAMIS_FILENAME = "c:\\foldingdata
       \\save_plan\\plan_for_origamis %d.csv";
23
24     FileObj[] fileObjs;
25     Plan[] plans;
26     MonitorOfPlanGroupOfPlanForOrigamis monitor;
27   public SimulatorToFindOptimalOrigamisWithInverting() {
28
29   }
30
31   void run() {
32     int i;
33     int j;
34
35     int numberOfPlansForOrigamis;
36     int optimalPlanForOrigamis;
37     int numberOfActiveEdgeOfOptimalPlanForOrigamis;
38
39     // Initiation
40     fileObjs =  new FileObj[NUMBER_OF_PLAN_FILES];
41     plans =    new PlanForOrigami[NUMBER_OF_PLAN_FILES];
42     monitor = new MonitorOfPlanGroupOfPlanForOrigamis();
43
44     // loads
45     for (i=0; i < NUMBER_OF_PLAN_FILES; i++) {
46       plans[i] = new PlanForOrigami();
47       ((PlanForOrigami) plans[i]).load(PLAN_FILENAME[i]);
48     }
49
50     numberOfPlansForOrigamis = 1 << NUMBER_OF_PLAN_FILES;
51
52     // 0, 0 invert, 90, 90 invert 180, 180 invert, 270 invert,  270
53
54     Planner [] planers = new Planner[numberOfPlansForOrigamis ];
55     for (i=0; i < numberOfPlansForOrigamis; i++ ) {
56       planers[i] = new PlanerForOrigamis(); // new planer
57
58       for (j=0; j < NUMBER_OF_PLAN_FILES; j++) {
59         if ( ( i & (1 << j)) != 0 ) {
60           if ( ( (PlanForOrigami) plans[j]).isInvert()) {
61           } else {
62             ( (PlanForOrigami) plans[j]).invert();
63           }
64         } else {
65           if (( (PlanForOrigami) plans[j]).isInvert()) {
66             ((PlanForOrigami) plans[j]).invert();
67           } else {
68
69           }
70         }
71       }
72       planers[i].build(plans);
73       planers[i].exportPlan(String.format(PLAN_FOR_ORIGAMIS_FILENAME, i));
74     }
```

```
75
76      optimalPlanForOrigamis = 0;
77      numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis) planers
            [0].getPlan()).getNumberOfActiveEdgesInPlanGroup();
78      for (i = 1; i< numberOfPlansForOrigamis; i++) {
79        if (numberOfActiveEdgeOfOptimalPlanForOrigamis > ((PlanForOrigamis)
              planers[i].getPlan()).getNumberOfActiveEdgesInPlanGroup()) {
80          optimalPlanForOrigamis = i;
81          numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis)
                planers[i].getPlan()).getNumberOfActiveEdgesInPlanGroup();
82        }
83        System.out.printf("planForOrigamis \t %d %d %d %f %f \n", i
84            , ((PlanForOrigamis) planers[i].getPlan()).
                getNumberOfActiveEdgesInPlanGroup()
85            , ((PlanForOrigamis) planers[i].getPlan()).
                getNumberOfActiveEdgesInPlansForOrigami()
86            , (float)((PlanForOrigamis) planers[i].getPlan()).
                getNumberOfActiveEdgesInPlanGroup() / (float) ( (
                PlanForOrigamis) planers[i].getPlan()).
                getNumberOfActiveEdgesInPlansForOrigami()
87            , (float)((PlanForOrigamis) planers[i].getPlan()).
                getNumberOfActiveEdgesInPlanGroup() / (float)( ((
                PlanForOrigamis) planers[i].getPlan()).getNumberOfEdges() *
                NUMBER_OF_PLAN_FILES));
88      }
89
90      System.out.printf("optimalPlanForOrigamis \t %d %d %d %f %f\n",
            optimalPlanForOrigamis
91              , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                    .getNumberOfActiveEdgesInPlanGroup()
92              , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                    .getNumberOfActiveEdgesInPlansForOrigami()
93              , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
                    getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)((
                    PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan()
                    ).getNumberOfActiveEdgesInPlansForOrigami()
94              , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
                    getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)
                    (((PlanForOrigamis) planers[optimalPlanForOrigamis].
                    getPlan()).getNumberOfEdges() * NUMBER_OF_PLAN_FILES));
95
96      //monitor.run((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan
            ());
97      monitor.run((PlanForOrigamis) planers[2].getPlan());
98    }
99
100   public static void main(String[] args) {
101     SimulatorToFindOptimalOrigamisWithInverting simulator = new
            SimulatorToFindOptimalOrigamisWithInverting ();
102     simulator.run();
103   }
104 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamis;
2
3  import com.drancom.programmableMatter.folding.dataFile.FileObj;
4  import com.drancom.programmableMatter.folding.monitor.
       MonitorOfPlanGroupOfPlanForOrigamis;
5  import com.drancom.programmableMatter.folding.origami.planner.Plan;
6  import com.drancom.programmableMatter.folding.origami.planner.Planner;
7  import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
       .PlanForOrigami;
8
9  public class SimulatorForOrigamis {
10
11   public static final String PLAN_FILENAME[] = {
12     "c:\\foldingdata\\save_8x8airplain\\plan_for_origami_8x8airplain.csv"
13     , "c:\\foldingdata\\save_8x8sailboat\\plan_for_origami_8x8sailboat.csv"
14     , "c:\\foldingdata\\save_8x8box\\plan_for_origami_8x8box.csv"
15     , "c:\\foldingdata\\save_8x8bench\\plan_for_origami_save_8x8bench.csv"
         };
16
17   public static final int NUMBER_OF_PLAN_FILES= 2;
18
19   public static final String PLAN_FOR_ORIGAMIS_FILENAME = "c:\\foldingdata
       \\save_plan\\plan_for_origamis %d.csv";
20
21   FileObj[] fileObjs;
22   Plan[] plans;
23   MonitorOfPlanGroupOfPlanForOrigamis monitor;
24
25   public SimulatorForOrigamis() {
26
27   }
28
29   void run() {
30     int i;
31     int j;
32
33     int numberOfPlansForOrigamis;
34     int optimalPlanForOrigamis;
35     boolean isThereAlinedOrigami;
36     int numberOfActiveEdgeOfOptimalPlanForOrigamis;
37
38     // Initiation
39     fileObjs =  new FileObj[NUMBER_OF_PLAN_FILES];
40     plans =   new PlanForOrigami[NUMBER_OF_PLAN_FILES];
41     monitor = new MonitorOfPlanGroupOfPlanForOrigamis();
42
43     // loads
44     for (i=0; i < NUMBER_OF_PLAN_FILES; i++) {
45       plans[i] = new PlanForOrigami();
46       ((PlanForOrigami) plans[i]).load(PLAN_FILENAME[i]);
47     }
48
49     numberOfPlansForOrigamis = 1 << NUMBER_OF_PLAN_FILES;
50
51     // 0, 0 invert, 90, 90 invert 180, 180 invert, 270 invert,  270
52
53     Planner [] planers = new Planner[numberOfPlansForOrigamis ];
54     for (i=0; i < numberOfPlansForOrigamis; i++ ) {
55       planers[i] = new PlanerForOrigamis(); // new planer
56
57       for (j=0; j < NUMBER_OF_PLAN_FILES; j++) {
58         if ( ( i & (1 << j)) != 0 ) {
59           if  ( ( (PlanForOrigami) plans[j]).isInvert()) {
60           } else {
61             ( (PlanForOrigami) plans[j]).invert();
62           }
63         } else {
64           if  ( ( ( PlanForOrigami ) plans[j] ).isInvert()) {
65             ( ( PlanForOrigami ) plans[j] ).invert();
66           } else {
67
68           }
69         }
70       }
71       planers[i].build(plans);
72       planers[i].exportPlan(String.format(PLAN_FOR_ORIGAMIS_FILENAME, i));
73     }
74
75     optimalPlanForOrigamis = 0;
76     numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis) planers
           [0].getPlan()).getNumberOfActiveEdgesInPlanGroup();
77
78     isThereAlinedOrigami  = false;
79
```

```
80      for (i = 1; i< numberOfPlansForOrigamis; i++) {
81        if (((PlanForOrigamis) planers[i].getPlan()).isAlined()) {
82          isThereAlinedOrigami  = true;
83          // finding plan for alined origamis
84          if (numberOfActiveEdgeOfOptimalPlanForOrigamis > ((PlanForOrigamis)
                planers[i].getPlan()).getNumberOfActiveEdgesInPlanGroup()) {
85            optimalPlanForOrigamis = i;
86            numberOfActiveEdgeOfOptimalPlanForOrigamis = ((PlanForOrigamis)
                  planers[i].getPlan()).getNumberOfActiveEdgesInPlanGroup();
87          }
88        }
89
90        System.out.printf("planForOrigamis \t %d %d %d %f %f ", i
91            , ((PlanForOrigamis) planers[i].getPlan()).
                  getNumberOfActiveEdgesInPlanGroup()
92            , ((PlanForOrigamis) planers[i].getPlan()).
                  getNumberOfActiveEdgesInPlansForOrigami()
93            , (float) ((PlanForOrigamis) planers[i].getPlan()).
                  getNumberOfActiveEdgesInPlanGroup() / (float) ( (
                  PlanForOrigamis) planers[i].getPlan()).
                  getNumberOfActiveEdgesInPlansForOrigami()
94            , (float) ((PlanForOrigamis) planers[i].getPlan()).
                  getNumberOfActiveEdgesInPlanGroup() / (float)( ((
                  PlanForOrigamis) planers[i].getPlan()).getNumberOfEdges() *
                  NUMBER_OF_PLAN_FILES));
95        if(((PlanForOrigamis) planers[i].getPlan()).isAlined()){
96          System.out.printf("true\n");
97          } else {
98            System.out.printf("false\n");
99          }
100     }
101
102     if (isThereAlinedOrigami  == true ) {
103       System.out.print("There is alined plen of origami");
104
105       System.out.printf("optimalPlanForOrigamis \t %d %d %d %f %f\n",
              optimalPlanForOrigamis
106             , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                    .getNumberOfActiveEdgesInPlanGroup()
107             , ((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan())
                    .getNumberOfActiveEdgesInPlansForOrigami()
108             , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
                    getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)((
                    PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan()
                    ).getNumberOfActiveEdgesInPlansForOrigami()
109             , (float)((PlanForOrigamis) planers[optimalPlanForOrigamis].
                    getPlan()).getNumberOfActiveEdgesInPlanGroup() / (float)
                    (((PlanForOrigamis) planers[optimalPlanForOrigamis].
                    getPlan()).getNumberOfEdges() * NUMBER_OF_PLAN_FILES));
110
111     } else {
112       System.out.print("There is not alined plan origamis  ");
113     }
114
115     monitor.run((PlanForOrigamis) planers[optimalPlanForOrigamis].getPlan()
            );
116
117     for(i=1;i<Math.pow(2, NUMBER_OF_PLAN_FILES);i++){
118       monitor.run((PlanForOrigamis) planers[i].getPlan());
119     }
120   }
121
122   public static void main(String[] args) {
123     SimulatorForOrigamis simulator = new SimulatorForOrigamis ();
124     simulator.run();
125   }
126 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamis;
2
3  import java.io.BufferedWriter;
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.io.FileOutputStream;
7  import java.io.FileWriter;
8  import java.io.IOException;
9
10 import com.drancom.programmableMatter.folding.controller.paper.Paper;
11 import com.drancom.programmableMatter.folding.dataFile.FilePlan;
12 import com.drancom.programmableMatter.folding.origami.planner.Plan;
13 import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
       .PlanForOrigami;
14
15 public class FilePlanForOrigamis implements FilePlan {
16
17   @Override
18   public void build(String fileName, Plan plan) {
19     int i;
20     int j;
21     int k;
22     int l;
23
24     boolean ready;
25
26     int numberOfGroups;
27     int numberOfEdges;
28     int numberOfOrigamis;
29     int numberOfPhases[];
30
31     File file = new File(fileName);
32     PlanForOrigamis planForOrigamis = (PlanForOrigamis) plan;
33
34     int [][][][]  planTable   = planForOrigamis.getPlanTable();
35     int [][][]   groupTable  = planForOrigamis.getPlanGroupTable();
36     float [][][]  edgeTable   = planForOrigamis.getEdgeTable();
37
38     numberOfGroups  = planForOrigamis.getNumberOfGroups();
39     numberOfEdges   = planForOrigamis.getNumberOfEdges();
40     numberOfOrigamis= planForOrigamis.getNumberOfOrigamis();
41     numberOfPhases  = planForOrigamis.getNumberOfPhases();
42
43     try {
44       boolean success = file.createNewFile();
45       if (success) {
46         // File did not exist and was created
47       } else {
48
49       }
50
51     } catch (IOException e) {
52       // TODO Auto-generated catch block
53       e.printStackTrace();
54     }
55
56     String bufferLine  = new String();
57
58     try {
59       BufferedWriter bufferedWriter = new BufferedWriter (new FileWriter (
             file));
60
61       bufferLine = String.format("# %d activeEdgesInPlansForOrigami",
             planForOrigamis.getNumberOfActiveEdgesInPlansForOrigami());
62       bufferedWriter.write(bufferLine);
63       bufferedWriter.newLine();
64
65       bufferLine = String.format("# %d activeEdgesInPlanGroup",
```

```
        planForOrigamis.getNumberOfActiveEdgesInPlanGroup());
66      bufferedWriter.write(bufferLine);
67      bufferedWriter.newLine();
68
69      bufferLine = String.format("# %f activeEdgesInPlanGroup /
            activeEdgesInPlansForOrigami", ((float)planForOrigamis.
            getNumberOfActiveEdgesInPlanGroup() / (float) planForOrigamis.
            getNumberOfActiveEdgesInPlansForOrigami()));
70      bufferedWriter.write(bufferLine);
71      bufferedWriter.newLine();
72
73      bufferLine = String.format("# %f activeEdgesInPlanGroup / (
            numberOfEdges * numberOfGroups)", (float) planForOrigamis.
            getNumberOfActiveEdgesInPlanGroup() / (float) (numberOfEdges *
            numberOfGroups));
74      bufferedWriter.write(bufferLine);
75      bufferedWriter.newLine();
76
77      bufferLine = String.format("# %d edges", numberOfEdges); // "# %d
            phases"
78      bufferedWriter.write(bufferLine);
79      bufferedWriter.newLine();
80
81      bufferLine = String.format("# startPointX startPointY endPointX
            endPointX "); // "# %d phases"
82      bufferedWriter.write(bufferLine);
83      bufferedWriter.newLine();
84
85      // print out to file
86      for (i = 0; i <numberOfEdges; i++) {
87        bufferLine = String.format("e");
88        for (j=0 ; j < 4 ; j++) {
89          bufferLine += String.format(", %f", edgeTable[i][j]);
90        }
91
92        // print out to file
93        bufferedWriter.write(bufferLine);
94        bufferedWriter.newLine();
95      }
96
97      // print group
98      bufferLine = String.format("# %d groups", numberOfGroups); // "# %d
            phases"
99      bufferedWriter.write(bufferLine);
100     bufferedWriter.newLine();
101
102     bufferLine = String.format("# groupNumber groupdata"); // "# %d
            phases"
103     bufferedWriter.write(bufferLine);
104     bufferedWriter.newLine();
105
106     // planGroup [groupNumber][edgeNumber]
107     for (i = 0; i < numberOfGroups; i++) {
108       for (j = 0; j < numberOfEdges; j++) {
109         bufferLine = String.format("g");
110         bufferLine += String.format(", %d, %d", i, groupTable[i][j]);
111
112         // print to file
113         bufferedWriter.write(bufferLine);
114         bufferedWriter.newLine();
115
116       }
117
118
119     }
120
121     // print origami plan
122     bufferLine = String.format("# %d origamis", numberOfOrigamis); // "#
            %d phases"
123     bufferedWriter.write(bufferLine);
124     bufferedWriter.newLine();
125
126
127     // planTable [phases][edgeNumber]
128     for (i = 0; i < numberOfOrigamis; i++) {
129       bufferLine = String.format("# %d phases", numberOfPhases[i]); // "#
              %d phases"
130       bufferedWriter.write(bufferLine);
131       bufferedWriter.newLine();
132
133       bufferLine = String.format("# origamis phases planData"); // "# %d
              phases"
134       bufferedWriter.write(bufferLine);
135       bufferedWriter.newLine();
136
137       for (j = 0; j < numberOfPhases[i]; j++) {
138         for (k = 0; k < numberOfGroups; k++) {
139           bufferLine = String.format("p");
140           bufferLine += String.format(", %d, %d, %d", i, j, planTable[i][
                j][k]);
141
142
143           // print to file
144           bufferedWriter.write(bufferLine);
145           bufferedWriter.newLine();
146
147         }
148       }
149     }
150
151     // file close
152
153     bufferedWriter.close();
154   } catch (FileNotFoundException e) {
155     e.printStackTrace();
156   } catch (IOException e) {
157     e.printStackTrace();
158   }
159 }
160
161 @Override
162 public void build(String fileName, Paper[] papers) {
163   // TODO Auto-generated method stub
164
165 }
166
167 @Override
168 public void read(String fileName, Plan plan) {
169 }
170
171 @Override
172 public void build(String fileName, Plan plan, Paper[] papers) {
173
174 }
175
176 @Override
177 public void read(String fileName, Paper[] papers) {
178   // TODO Auto-generated method stub
179
180 }
181
182 @Override
183 public void read(String fileName, Plan plan, Paper[] papers) {
184   // TODO Auto-generated method stub
185
186 }
187
188 }
```

```java
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;
        }
      }
```

```
262       searchMode = 0;
263       while (!isNo0Tag) {
264         if(isNoMoreEdgeToTag == true){
265           searchMode = 0;
266         }
267
268         isNoMoreEdgeToTag = true;
269         for (j=0;j<numberOfEdges;j++) {
270           if(planGroup[i][j] != 0) {
271
272             x0 = (int)((edgeTable[j][0])* 8);
273             y0 = (int)(edgeTable[j][1]* 8);
274             x1 = (int)((edgeTable[j][2])* 8);
275             y1 = (int)(edgeTable[j][3]* 8);
276
277             if( x0 <0 ) {
278               x0 = (int)((edgeTable[j][0]+1)* 8);
279             }
280             if( x1 <0 ) {
281               x1 = (int)((edgeTable[j][2]+1)* 8);
282             }
283             if (searchMode == 0 ){
284               if(vertexs[i][x0][y0][3] == 0
285                   && vertexs[i][x0][y0][2] == 1
286                   && vertexs[i][x1][y1][3] == 0
287                   && vertexs[i][x1][y1][2] == 1 ){
288                 vertexs[i][x0][y0][3] = 1;
289                 vertexs[i][x1][y1][3] = 2;
290
291
292               } else if(vertexs[i][x0][y0][3] == 0
293                   && vertexs[i][x0][y0][2] == 1
294                   && vertexs[i][x1][y1][3] == 0
295                   && vertexs[i][x1][y1][2] > 1 ){
296                 vertexs[i][x0][y0][3] = 1;
297                 vertexs[i][x1][y1][3] = 3;
298
299                 searchMode = 1;
300                 isNoMoreEdgeToTag = false;
301
302               } else if(vertexs[i][x0][y0][3] == 0
303                   && vertexs[i][x0][y0][2] > 1
304                   && vertexs[i][x1][y1][3] == 0
305                   && vertexs[i][x1][y1][2] == 1 ){
306                 vertexs[i][x0][y0][3] = 3;
307                 vertexs[i][x1][y1][3] = 1;
308
309                 searchMode = 1;
310                 isNoMoreEdgeToTag = false;
311
312               }
313             }else if ( searchMode == 1) {
314
315               if(vertexs[i][x0][y0][3] == 3
316                 && vertexs[i][x0][y0][2] > 1
317                 && vertexs[i][x1][y1][3] == 0
318                 && vertexs[i][x1][y1][2] > 1 ){
319                 vertexs[i][x1][y1][3] = 3;
320                 isNoMoreEdgeToTag = false;
321
322               } else if(vertexs[i][x0][y0][3] == 0
323                   && vertexs[i][x0][y0][2] > 1
324                   && vertexs[i][x1][y1][3] == 3
325                   && vertexs[i][x1][y1][2] > 1 ){
326                 vertexs[i][x0][y0][3] = 3;
327                 isNoMoreEdgeToTag = false;
328
329               } else if(vertexs[i][x0][y0][3] == 3
330                   && vertexs[i][x0][y0][2] >= 1
331                   && vertexs[i][x1][y1][3] == 0
332                   && vertexs[i][x1][y1][2] == 1 ){
333                 vertexs[i][x1][y1][3] = 2;
334
335                 isNoMoreEdgeToTag = false;
336
337               } else if(vertexs[i][x0][y0][3] == 0
338                   && vertexs[i][x0][y0][2] == 1
339                   && vertexs[i][x1][y1][3] == 3
340                   && vertexs[i][x1][y1][2] >= 1 ){
341                 vertexs[i][x0][y0][3] = 2;
342                 isNoMoreEdgeToTag = false;
343
344
345               }
346             }
347           }
348         }
349         if(isNoMoreEdgeToTag == true && searchMode == 0){
350           for (j=0;j<numberOfEdges;j++) {
351             if(planGroup[i][j] != 0) {
352               if(planGroup[i][j] != 0) {
353
354
355                 x0 = (int)((edgeTable[j][0])* 8);
356                 y0 = (int)(edgeTable[j][1]* 8);
357                 x1 = (int)((edgeTable[j][2])* 8);
358                 y1 = (int)(edgeTable[j][3]* 8);
359
360
361                 if( x0 <0 ) {
362                   x0 = (int)((edgeTable[j][0]+1)* 8);
363                 }
364                 if( x1 <0 ) {
365                   x1 = (int)((edgeTable[j][2]+1)* 8);
366                 }
367
368                 if(searchMode == 0
369                   && vertexs[i][x0][y0][3] == 0
370                   && vertexs[i][x0][y0][2] >= 1
371                   && vertexs[i][x1][y1][3] == 0
372                   && vertexs[i][x1][y1][2] >= 1 ){
373
374                   vertexs[i][x0][y0][3] = 2;
375                   vertexs[i][x1][y1][3] = 3;
376                   isNoMoreEdgeToTag = false;
377
378                   searchMode = 1;
379                 }
380               }
381             }
382           }
383         }
384
385         isNo0Tag = true;
386         for (j = 0; j < 9; j++) {
387           for (k = 0; k < 9; k++) {
388             if (vertexs[i][j][k][3] == 0){
389               isNo0Tag = false;
390               break;
391             }
392           }
393         }
394       }
395     }
396
397
398
399     // 1. find a vertex connected nothing active edge. put tag 4 on it
```

```
400       // 2. find a vertex connected one active edge. put tag 1 on it.
401       // 3. find a vertex connected tag 1 or 3 vertex through active edge .
402       //     if it is not a dead end put tag 3 on. if it is dead end put tag 2
403       // 4. go to 2
404       // 5. go to 1
405
406
407
408       // build plan
409       planForOrigamis = new PlanForOrigamis();
410
411       planForOrigamis.setPlanTable(planTableForOrigamis);
412       planForOrigamis.setPlanGroupTable(planGroup);
413       planForOrigamis.setEdgeTable(plansForOrigami[0].getEdgeTable());
414
415       planForOrigamis.setNumberOfOrigamis(numberOfOrigamis);
416       planForOrigamis.setNumberOfEdges(numberOfEdges);
417       planForOrigamis.setNumberOfPhases(lastPhase);
418       planForOrigamis.setNumberOfGroups(numberOfGroups);
419
420       planForOrigamis.setNumberOfActiveEdgesInPlansForOrigami(
              numberOfActiveEdgesInPlansForOrigami);
421
422       // Vertex
423       planForOrigamis.setVertexs(vertexs);
424     }
425
426     @Override
427     public void exportPlan(String fileName) {
428       planForOrigamis.export(fileName);
429     }
430
431     @Override
432     public void exportPlan(String fileName, Paper[] papers) {
433       FilePlan filePlan = new FilePlan ForOrigamis();
434
435       filePlan.build(fileName, planForOrigamis);
436     }
437
438     @Override
439     public Plan getPlan() {
440       return this.planForOrigamis;
441     }
442
443 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamis;
2
3  import com.drancom.programmableMatter.folding.dataFile.FilePlan;
4  import com.drancom.programmableMatter.folding.origami.planner.Plan;
5
6  public class PlanForOrigamis implements Plan {
7    float edgeTable[][];   // [edge number] [x0 y0 x1 y2]
8    int planTable[][][][];   // [numberOfOrigamis] [numberOfPhases] [
         numberOfGroups]
9    int planGroup[][];      // [numberOfGroups] [numberOfEdges]
10
11   float vertexs[][][][][]; // [numberOfGroups][x][y][0:x position, 1: y
         position, 2: NumberOfConnectedActiveEdges, 3:tag]
12
13   int numberOfEdges;
14   int numberOfPhases[];
15   int numberOfGroups;
16   int numberOfOrigamis;
17   int numberOfActiveEdgesInPlansForOrigami;
18
19   public boolean isAlined() {
20
21     int i;
22     int j;
23
24     boolean isFoldingOneway;
25
26     for (i = 0; i < getNumberOfEdges(); i++) {
27       isFoldingOneway = false;
28       for (j = 0; j < getNumberOfGroups(); j++) {
29         if(planGroup[j][i] != 0) {
30           if (isFoldingOneway == true){
31             return false;
32           } else {
33             isFoldingOneway = true;
34           }
35         }
36       }
37     }
38     return true;
39   }
40
41   public int [][] getConbinedPlanMap() {
42     int i;
43     int j;
44
45     int [][] conbinedPlanMap = new int[numberOfGroups][numberOfEdges];
46
47     for ( i = 0 ; i < numberOfEdges ; i++) {
48       for (j = 0 ; j < numberOfGroups ; j++) {
49         conbinedPlanMap[j][i] = 0;
50       }
51     }
52
53
54     for ( i = 0 ; i < numberOfEdges ; i++) {
55       for (j = 0 ; j < numberOfGroups ; j++) {
56         if (planGroup[j][i] == 1) {
57           if (conbinedPlanMap[j][i] != 0) {
58             conbinedPlanMap[j][i] = 1;
59           } else {
60             conbinedPlanMap[j][i] = 2;
61           }
62         } else if (planGroup[j][i] == -1) {
63           if (conbinedPlanMap[j][i] != 0) {
64             conbinedPlanMap[j][i] = 1;
65           } else {
66             conbinedPlanMap[j][i] = -1;
67           }
68         }
69       }
70     }
71
72
73     return conbinedPlanMap;
74
75   }
76
77   public void setNumberOfOrigamis(int numberOfOrigamis) {
78     this.numberOfOrigamis = numberOfOrigamis;
79   }
80
81   public int getNumberOfOrigamis(){
82     return numberOfOrigamis;
83   }
84
85   public int[][][][] getPlanTable(){
86     return planTable;
87   }
88
89   public void setPlanTable(int[][][][] planTable){
```

```
90      int i;
91      int j;
92      int k;
93
94      this.planTable = new int [planTable.length][planTable[0].length][
            planTable[0][0].length];
95
96      for (i=0; i<planTable.length ; i++) {
97        for(j=0; j< planTable[i].length ; j++) {
98          for(k=0; k< planTable[i][j].length ; k++) {
99            this.planTable[i][j][k] = planTable[i][j][k];
100         }
101       }
102     }
103   }
104
105   public int[][] getPlanGroupTable(){
106     return planGroup;
107   }
108
109   public void setPlanGroupTable(int [][] planGroup) {
110     int i;
111     int j;
112
113     this.planGroup = new int [planGroup.length][planGroup[0].length];
114
115     for (i=0; i<planGroup.length ; i++) {
116       for(j=0; j< planGroup[i].length ; j++) {
117         this.planGroup[i][j] = planGroup[i][j];
118       }
119     }
120   }
121
122   public void setEdgeTable(float[][] edgeTable){
123     int i;
124     int j;
125
126     this.edgeTable = new float [edgeTable.length][edgeTable[0].length];
127
128     for (i=0; i<edgeTable.length ; i++) {
129       for(j=0; j< edgeTable[i].length ; j++) {
130         this.edgeTable[i][j] = edgeTable[i][j];
131       }
132     }
133   }
134
135   public float[][] getEdgeTable(){
136     return edgeTable;
137   }
138
139   public float[][][][][] getVertexs() {
140     return vertexs;
141   }
142
143   public void setVertexs(float[][][][][] vertexs) {
144     this.vertexs = vertexs;
145   }
146
147   public int[] getNumberOfPhases() {
148     return numberOfPhases;
149   }
150
151   public void setNumberOfPhases(int[] numberOfPhases) {
152     int i;
153
154     this.numberOfPhases = new int [numberOfPhases.length];
155
156     for (i=0; i<numberOfPhases.length ; i++) {
157       this.numberOfPhases[i] = numberOfPhases[i];
158     }
159   }
160
161   public int getNumberOfEdges() {
162     return numberOfEdges;
163   }
164
165   public void setNumberOfEdges(int numberOfEdges) {
166     this.numberOfEdges = numberOfEdges;
167   }
168
169   public void export (String fileName) {
170     FilePlan filePlan = new FilePlanForOrigamis();
171     filePlan.build(fileName, this);
172   }
173
174   public void load(String fileName){
175     FilePlan filePlan = new FilePlanForOrigamis();
176     filePlan.read(fileName, this);
177   }
178
179   public int getNumberOfGroups() {
180     return numberOfGroups;
181   }
182
183   public void setNumberOfGroups(int numberOfGroups) {
184     this.numberOfGroups = numberOfGroups;
185   }
186
187   public int getNumberOfActiveEdgesInPlanGroup() {
188     int i;
189     int j;
190
191     int numberOfActiveEdges = 0;
192
193     for (i=0; i < numberOfGroups ; i++) {
194       for (j=0; j < numberOfEdges; j++) {
195         if (planGroup[i][j] != 0) {
196           numberOfActiveEdges++;
197         }
198       }
199     }
200     return numberOfActiveEdges;
201   }
202
203   public int getNumberOfActivePlanGroup() {
204     int i;
205     int j;
206
207     int numberOfActivePlanGroup = 0;
208
209     for (i = 0 ; i < numberOfGroups ; i++) {
210       for (j = 0 ; j < numberOfEdges ; j++) {
211         if (planGroup[i][j] != 0) {
212           numberOfActivePlanGroup++;
213           break;
214         }
215       }
216     }
217     return numberOfActivePlanGroup;
218   }
219
220   public void setNumberOfActiveEdgesInPlansForOrigami(int
          numberOfActiveEdgesInPlansForOrigami) {
221     this.numberOfActiveEdgesInPlansForOrigami =
            numberOfActiveEdgesInPlansForOrigami;
222   }
223
224   public int getNumberOfActiveEdgesInPlansForOrigami() {
```

```
225     return numberOfActiveEdgesInPlansForOrigami;
226   }
227 }
```

```
1  package com.drancom.programmableMatter.folding.controller.paper;
2
3  import java.awt.List;
4  import java.util.ArrayList;
5
6  import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
7  import com.drancom.programmableMatter.folding.monitor.MonitorOfPaperArray;
8  import com.sun.corba.se.spi.legacy.connection.GetEndPointInfoAgainException
       ;
9  import com.sun.org.apache.bcel.internal.generic.NEWARRAY;
10 import com.sun.xml.internal.bind.marshaller.MinimumEscapeHandler;
11
12 public class Paper {
13
14   ArrayList<Paper> papersForMonitor;
15   MonitorOfPaperArray monitor = new MonitorOfPaperArray();
16
17   public final static int MAX_NUMBER_OF_POINTS = 1000;
18   public final static int MAX_NUMBER_OF_POLYGONS = 1000;
19   public final static int MAX_NUMBER_OF_LINES = 1000;
20
21   public final static double ERROR_RATIO_FOR_LANGTH = 0.05f;
22   public final static double MAX_NUMBER_OF_LINKAGE_EFFECT = 0.01f;
23
24   Point[] pointsOnPaper;
25   int numberOfPoints = 0;
26   Line[] lines;
27   protected int numberOfLines = 0;
28   Polygon[] polygons;
29   int numberOfPolygons;
30
31   Point sortedPointsMetrix[][];
32   Point sortedPointsLine[];
33
34   double barEnergyOfPaper = 0.0f;
35
36   public Paper() {
37     pointsOnPaper = new Point[MAX_NUMBER_OF_POINTS];
38     lines = new Line[MAX_NUMBER_OF_LINES];
39     polygons = new Polygon[MAX_NUMBER_OF_POLYGONS];
40     // numberOfPoints;
41     // numberOfLines;
42   }
43
44   // get the values
45   public Point getPoint(int index) {
46     return pointsOnPaper[index];
47   }
48
49   public Point getPoint(double x, double y, double z) {
50     int i;
51     for (i = 0; i < getNumberOfPoints(); i++) {
52       if (pointsOnPaper[i].getXOnPaper() == x && pointsOnPaper[i].
             getYOnPaper() == y
53           && pointsOnPaper[i].getZOnPaper() == z) {
54         return pointsOnPaper[i];
55
56       }
57     }
58     return null;
59   }
60
61   public Point[] getPoints() {
62     return pointsOnPaper;
63   }
```

```
64
65   public Line[] getLines() {
66     return lines;
67   }
68
69   public Polygon[] getPolygons() {
70     return polygons;
71   }
72
73   public Line getLine(int index) {
74     return lines[index];
75   }
76
77   public int getNumberOfPolygons() {
78     return numberOfPolygons;
79   }
80
81   public int getNumberOfColoums() {
82     return (int) Math.sqrt((double) numberOfPoints);
83   }
84
85   // set the values
86   public void setPoint(int index, Point point) {
87     if (index > MAX_NUMBER_OF_POINTS) {
88       System.err.print("over MAX_NUMBER_OF_POINTS");
89     }
90
91     this.pointsOnPaper[index] = point;
92
93     if (index > numberOfPoints - 1) {
94       numberOfPoints = index + 1;
95     }
96   }
97
98   public void setLine(int index, Line line) {
99     if (index > MAX_NUMBER_OF_LINES) {
100      System.err.printf("over MAX_NUMBER_OF_LINES");
101    }
102
103    this.lines[index] = line;
104
105    if (index > numberOfLines - 1) {
106      numberOfLines = index + 1;
107    }
108  }
109
110  // build paper
111  public void build() {
112    // reset
113    copyFromCoordinationOnPaperToCoordinationInReal();
114
115    // Sorting Points
116    buildSortedPoints();
117
118    // set up Groups of Poligons
119    buildPolygons();
120
121    buildPointsByAnglesOfLines();
122  }
123
124  // sort line
125  public void sortLine() {
126    Line sortedLines[] = new Line[numberOfLines];
127    Line tempLine;
128
129    int i;
130    int j;
131
132    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
```

```
399                 .getType() == Line.TYPE_STATIC_LINE)) {
400
401       } else {
402         return false;
403       }
404
405       axisVector = new Vector();
406       // get axisVector
407       // axisVector = axisLine.getVector();
408       axisVector.setXYZ(axisLinePoint1.getXInReal()
409           - axisLinePoint2.getXInReal(), axisLinePoint1.getYInReal()
410           - axisLinePoint2.getYInReal(), axisLinePoint1.getZInReal()
411           - axisLinePoint2.getZInReal());
412
413       // get counterpartVector
414       counterpartVector = new Vector();
415       counterpartVector.setXYZ(counterpartPoint.getXInReal()
416           - originVector.getX(), counterpartPoint.getYInReal()
417           - originVector.getY(), counterpartPoint.getZInReal()
418           - originVector.getZ());
419
420       // get newVector
421       newVector1 = new Vector();
422       newVector2 = new Vector();
423       newVector3 = new Vector();
424
425       newVector1.setXYZ(counterpartVector);
426       newVector3.setXYZ(counterpartVector);
427
428       newVector1.rotation(axisVector, (float) ( Math.PI - axisLine.getAngle()
              ));
429       newVector3.invert();
430
431       newVector = newVector1;
432
433       if (axisLine.getType() == Line.TYPE_STATIC_LINE) {
434         newVector = newVector3;
435       }
436
437       newVector.transform(originVector);
438
439       // transform to originVector
440       updatePoint.setVectorInReal(newVector);
441
442       // isRenewed
443       updatePoint.isRenewed = true;
444       return true;
445     }
446
447     void transform(Vector vector) {
448       transform(vector.getX(), vector.getY(), vector.getZ());
449     }
450
451     // transform or rotation whole paper
452     void transform(float x, float y, float z) {
453       int i;
454       for (i = 0; i < numberOfPoints; i++) {
455         pointsOnPaper[i].setPointInReal(pointsOnPaper[i].getXInReal() + x,
                pointsOnPaper[i]
456               .getYInReal()
457             + y, pointsOnPaper[i].getZInReal() + z);
458       }
459     }
460
461     void rotation(Vector axisVector, float angle) {
462       int i;
463       Vector tempVector;
464       for (i = 0; i < numberOfPoints; i++) {
465
466         // -1 * theta rotation by z axis
467
468         tempVector = pointsOnPaper[i].getVectorInReal();
469         tempVector.rotation(axisVector, angle);
470         pointsOnPaper[i].setVectorInReal(tempVector);
471       }
472
473     }
474
475     void rotation(float r, float theta, float phi, float angle) {
476
477       Vector axisVector = new Vector();
478
479       axisVector.setRThetaPhi(r, theta, phi);
480
481       rotation(axisVector, angle);
482     }
483
484     public Line getLine(Point point0, Point point1) {
485
486       int i;
487
488       for (i = 0; i < numberOfLines; i++) {
489         if ((point0 == lines[i].getStartPoint() && point1 == lines[i]
490             .getEndPoint())
491             || (point1 == lines[i].getStartPoint() && point0 == lines[i]
492                 .getEndPoint())) {
493
494           return lines[i];
495         }
496       }
497       return null;
498     }
499
500     // set up sortedPoint
501     void buildSortedPoints() {
502       int i, j;
503       Point tempPoint;
504
505       int numberOfColumes = (int) Math.sqrt((double) numberOfPoints);
506
507       sortedPointsLine = new Point[numberOfPoints];
508
509       for (i = 0; i < numberOfPoints; i++) {
510         sortedPointsLine[i] = pointsOnPaper[i];
511       }
512
513       for (i = 0; i < numberOfPoints; i++) {
514         for (j = i + 1; j < numberOfPoints; j++) {
515           if ((sortedPointsLine[i].getYOnPaper() > sortedPointsLine[j]
516               .getYOnPaper())
517               || ((sortedPointsLine[i].getYOnPaper() == sortedPointsLine[j]
518                   .getYOnPaper()) && (sortedPointsLine[i]
519                   .getXOnPaper() >= sortedPointsLine[j]
520                   .getXOnPaper()))) {
521             tempPoint = sortedPointsLine[i];
522             sortedPointsLine[i] = sortedPointsLine[j];
523             sortedPointsLine[j] = tempPoint;
524           }
525         }
526       }
527
528       sortedPointsMetrix = new Point[numberOfColumes][numberOfColumes];
529
530       for (i = 0; i < numberOfPoints; i++) {
531         sortedPointsMetrix[i / numberOfColumes][i % numberOfColumes] =
              sortedPointsLine[i];
532       }
533     }
```

```
534
535   public int getNumberOfPoints() {
536     return numberOfPoints;
537   }
538
539   public int getNumberOfEdges() {
540     return numberOfLines;
541   }
542
543   public void move(Vector vector) {
544
545   }
546
547   public void rotate(float theta, float pi) {
548
549   }
550
551   // function for Polygons
552   void buildPolygons() {
553     int i;
554     int j;
555     int k;
556
557     numberOfPolygons = 0;
558
559     Point[] polygonPoints;
560
561     polygonPoints = new Point[3];
562
563     int numberOfColoums = getNumberOfColoums();
564
565     // build Polygons with clockwise
566
567     for (i = 0; i < numberOfPoints; i++) {
568       for (j = i + 1; j < numberOfPoints; j++) {
569         if (getLine(pointsOnPaper[i], pointsOnPaper[j]) != null) {
570           for (k = j + 1; k < numberOfPoints; k++) {
571             if (getLine(pointsOnPaper[j], pointsOnPaper[k]) != null
572                 && getLine(pointsOnPaper[k], pointsOnPaper[i]) != null) {
573
574               // initiation polygon Points
575
576               // set a polygon with Clockwise
577
578               // if two point's x on paper is same;
579               if (pointsOnPaper[j].getXOnPaper() == pointsOnPaper[k]
580                   .getXOnPaper()) {
581
582                 if (pointsOnPaper[i].getYOnPaper() < pointsOnPaper[j]
583                     .getYOnPaper()) {
584                   // pick a left point
585                   polygonPoints[0] = pointsOnPaper[i];
586
587                   // pick a up point
588                   if (pointsOnPaper[j].getYOnPaper() > pointsOnPaper[k]
589                       .getYOnPaper()) {
590
591                     polygonPoints[1] = pointsOnPaper[j];
592                     polygonPoints[2] = pointsOnPaper[k];
593
594                   } else {
595
596                     polygonPoints[1] = pointsOnPaper[k];
597                     polygonPoints[2] = pointsOnPaper[j];
598
599                   }
600                 } else {
601
602                   // pick a right point
603                   polygonPoints[0] = pointsOnPaper[i];
604
605                   // pick the down point
606                   if (pointsOnPaper[j].getYOnPaper() < pointsOnPaper[k]
607                       .getYOnPaper()) {
608
609                     polygonPoints[1] = pointsOnPaper[j];
610                     polygonPoints[2] = pointsOnPaper[k];
611
612                   } else {
613
614                     polygonPoints[1] = pointsOnPaper[k];
615                     polygonPoints[2] = pointsOnPaper[j];
616
617                   }
618                 }
619               } else if ((pointsOnPaper[i].getYOnPaper() - pointsOnPaper[j]
620                   .getYOnPaper())
621                   * (pointsOnPaper[j].getYOnPaper() - pointsOnPaper[k]
622                       .getYOnPaper()) < (pointsOnPaper[i]
623                   .getXOnPaper() - pointsOnPaper[j].getXOnPaper())
624                   * (pointsOnPaper[j].getXOnPaper() - pointsOnPaper[k]
625                       .getXOnPaper())) {
626
627                 // pick a left down point
628                 polygonPoints[0] = pointsOnPaper[i];
629
630                 if (pointsOnPaper[j].getXOnPaper() > pointsOnPaper[k]
631                     .getXOnPaper()) {
632
633                   // pick the up point
634                   polygonPoints[1] = pointsOnPaper[j];
635                   polygonPoints[2] = pointsOnPaper[k];
636
637                 } else {
638
639                   // pick the down point
640                   polygonPoints[1] = pointsOnPaper[k];
641                   polygonPoints[2] = pointsOnPaper[j];
642
643                 }
644
645               } else {
646                 // pick a right up point
647                 polygonPoints[0] = pointsOnPaper[i];
648
649                 if (pointsOnPaper[j].getYOnPaper() < pointsOnPaper[k]
650                     .getYOnPaper()) {
651
652                   // pick a down point
653                   polygonPoints[1] = pointsOnPaper[j];
654                   polygonPoints[2] = pointsOnPaper[k];
655
656                 } else {
657
658                   // pick a up point
659                   polygonPoints[1] = pointsOnPaper[k];
660                   polygonPoints[2] = pointsOnPaper[j];
661                 }
662               }
663               // make polygon
664               polygons[numberOfPolygons] = new Polygon();
665               try {
666                 polygons[numberOfPolygons].setPolygon(
667                     polygonPoints, this);
668               } catch (NoLineException e) {
669
670                 e.printStackTrace();
671               } catch (Exception e) {
```

```
672
673                 e.printStackTrace();
674               }
675               numberOfPolygons++;
676             }
677           }
678         }
679       }
680     }
681   }
682
683   public Polygon getPolygon(int index) {
684
685     if (index < 0 || index > numberOfPolygons) {
686       return null;
687     }
688
689     return polygons[index];
690
691   }
692
693   public Polygon getPolygon(Point[] point) throws Exception {
694     Point polygonPoints[];
695
696     int numberOfMatchPoints;
697
698     int i;
699     int j;
700     int k;
701
702     for (i = 0; i < numberOfPolygons; i++) {
703       polygonPoints = polygons[i].getPoints();
704       if (polygonPoints.length == point.length) {
705         for (j = 0; j < polygonPoints.length; j++) {
706           numberOfMatchPoints = 0;
707           for (k = 0; k < polygonPoints.length; k++) {
708             if (polygonPoints[k] != point[(k + j)
709                 % polygonPoints.length]) {
710               break;
711             }
712             numberOfMatchPoints++;
713           }
714           if (numberOfMatchPoints == polygonPoints.length) {
715             return polygons[i];
716           }
717         }
718       }
719     }
720     return null;
721   }
722
723   public Polygon[] getPolygons(Point point) {
724     Point[] pointArray = new Point[1];
725
726     pointArray[0] = point;
727
728     return getPolygons(pointArray);
729   }
730
731   public Polygon[] getPolygons(Point point0, Point point1) {
732     Point[] pointArray = new Point[2];
733
734     pointArray[0] = point0;
735     pointArray[1] = point1;
736
737     return getPolygons(pointArray);
738   }
739
740   public Polygon[] getPolygons(Point[] points) {
741     int i, j, k;
742     Polygon[] polygonsHavingPoints;
743     Polygon[] oldPolygonsHavingPoints;
744     int numberOfPolygonsHavingPoints;
745
746     polygonsHavingPoints = null;
747     numberOfPolygonsHavingPoints = 0;
748     for (i = 0; i < numberOfPolygons; i++) {
749       if (polygons[i].isAllPointsHave(points)) {
750
751         oldPolygonsHavingPoints = polygonsHavingPoints;
752         polygonsHavingPoints = new Polygon[numberOfPolygonsHavingPoints +
                1];
753
754         for (j = 0; j < numberOfPolygonsHavingPoints; j++) {
755           polygonsHavingPoints[j] = oldPolygonsHavingPoints[j];
756
757         }
758         polygonsHavingPoints[numberOfPolygonsHavingPoints] = polygons[i];
759         numberOfPolygonsHavingPoints++;
760
761       }
762     }
763     return polygonsHavingPoints;
764   }
765
766   public Polygon getCounterpartPolygon(Polygon polygon, Point point0,
767       Point point1) {
768     Polygon[] tempPolygons;
769     int i;
770     Point[] sharedPoints;
771
772     sharedPoints = new Point[2];
773
774     sharedPoints[0] = point0;
775     sharedPoints[1] = point1;
776
777     tempPolygons = getPolygons(sharedPoints);
778
779     for (i = 0; i < tempPolygons.length; i++) {
780       if (tempPolygons[i] != polygon) {
781         return tempPolygons[i];
782       }
783     }
784
785     return null;
786   }
787
788   public Point getCounterpartPointOnCounterpartPolygon(Polygon polygon,
789       Point point0, Point point1) {
790     Polygon counterpartPolygon;
791     int i;
792     Point[] pointsOfCounterpartPolygon;
793
794     counterpartPolygon = getCounterpartPolygon(polygon, point0, point1);
795
796     if (counterpartPolygon == null) {
797       return null;
798     }
799
800     pointsOfCounterpartPolygon = counterpartPolygon.getPoints();
801
802     for (i = 0; i < pointsOfCounterpartPolygon.length; i++) {
803
804       if ((pointsOfCounterpartPolygon[i] != point0)
805           && (pointsOfCounterpartPolygon[i] != point1)) {
806
807         return pointsOfCounterpartPolygon[i];
808       }
```

```
809
810       }
811
812       return null;
813     }
814
815     boolean changeAngle(Polygon polygon, Point point0, Point point1,
816         float angle) {
817
818       /**
819        * 1. if this line is edge line, false 2. get vector from the line
820        * changing angle to the oppositePoint from line. 2. rotate vector with
821        * angle.
822        */
823
824       int i;
825
826       Line line;
827       Point changedPoint;
828       Polygon counterpartPolygon;
829
830       Vector vector;
831       Vector oldVectorInRealOnChangedPoint;
832       Vector axisVector;
833       Vector originVector;
834
835       Vector va, vb, vc;
836
837       line = getLine(point0, point1);
838
839       if (line.getType() == Line.TYPE_EDGE_LINE) {
840         return false;
841       }
842
843       if (line.getAngle() == angle) {
844         return true;
845       }
846
847       // debug monitor
848
849       counterpartPolygon = getCounterpartPolygon(polygon, point0, point1);
850
851       changedPoint = getCounterpartPointOnCounterpartPolygon(polygon, point0,
852           point1);
853
854       Vector []  oldVectors = new Vector[getNumberOfPoints()];
855
856       for (i = 0 ; i<getNumberOfPoints(); i++) {
857         oldVectors[i] = pointsOnPaper[i].getVectorInReal();
858       }
859
860       originVector = counterpartPolygon.getOriginVector(changedPoint, point0,
861           point1);
862
863       vector = polygon.getVectorFromLineToOppositePoint(point0, point1);
864
865       // rotation
866       axisVector = polygon.getVectorOnTheLine(point0, point1);
867
868       vector.rotation(axisVector.getUnitVector(), ((float) Math.PI) - angle);
869
870       // translate to origin vector;
871
872       vector.addVector(originVector);
873
874       changePointInReal(changedPoint, vector);
875
876       if (fixLinkage(changedPoint)) {
877
878         return true;
879       } else {
880         for (i = 0 ; i<getNumberOfPoints(); i++) {
881           changePointInReal(pointsOnPaper[i], oldVectors[i]);
882         }
883         return false;
884       }
885     }
886
887     // adding an angle
888     boolean addAngle(Polygon polygon, Point point0, Point point1, float angle
889         ) {
890       int i;
891
892       Paper paper;
893       int polygonIndex;
894
895       Polygon counterpartPolygon;
896
897       Vector unitVectorOnLineOnCounterpatPolygon;
898       Vector vectorFromLineToOppositePoint;
899       Point oppsitePointOnCounterpatPolygon;
900
901       // error check;
902       Line line;
903       line = getLine(point0, point1);
904       if (line == null) {
905         return false;
906       }
907
908       if (line.getType() == Line.TYPE_EDGE_LINE || line.getAngle() == 0.0f) {
909         return true;
910       }
911
912       // get counterpartPolygon
913       counterpartPolygon = getCounterpartPolygon(polygon, point0, point1);
914       if (counterpartPolygon == null) {
915         return true;
916       }
917
918       // get unit Vector OnLine
919       unitVectorOnLineOnCounterpatPolygon = counterpartPolygon
920           .getVectorOnTheLine(point0, point1).getUnitVector();
921
922       // get Vector from line to opposite Point
923       vectorFromLineToOppositePoint = counterpartPolygon
924           .getVectorFromLineToOppositePoint(point0, point1);
925
926       // rotation the vector
927       vectorFromLineToOppositePoint.rotation(
928           unitVectorOnLineOnCounterpatPolygon, angle);
929
930       // translate
931       vectorFromLineToOppositePoint.transform(polygon
932           .getVectorOfStartPointFromLineToOppositePoint(point0, point1));
933
934       // reset the new vector
935       oppsitePointOnCounterpatPolygon =
936           getCounterpartPointOnCounterpartPolygon(
937           polygon, point0, point1);
938       oppsitePointOnCounterpatPolygon
939           .setVectorInReal(vectorFromLineToOppositePoint);
940
941       return fixLinkage(oppsitePointOnCounterpatPolygon);
942     }
943
944     public boolean changePointInReal(Point point, Vector vector) {
945
946       point.setPointInReal(vector.getX(), vector.getY(), vector.getZ());
```

```
945
946       // 1 0 0 -1 0 0
947       // check the angles
948
949       // is type of angle is different from line type.
950
951       return true;
952
953     }
954
955     public void resetAngleOnLineByPoints(Line line) {
956
957     }
958
959     boolean checkError(Point changedPoint) {
960       if (!checkPointError(changedPoint)) {
961         return false;
962       }
963       if (!checkAngleError(changedPoint)) {
964         return false;
965       }
966
967       return true;
968     }
969
970     boolean checkAngleError(Point changedPoint) {
971       return true;
972     }
973
974     boolean checkPointError(Point changedPoint) {
975       /*
976        * error check
977        *
978        * find polygon shared the changed point if there are two line change
               in
979        * one polygon return false.
980        *
981        * if not fix the linkage set point 0 = changedPoint set point 1 until
982        * there are no error -> return true;
983        *
984        * or return false and role back;
985        */
986
987       int i;
988       int j;
989       int k;
990
991       float lengthOfFirstLineOnPaper;
992       float lengthOfFirstLineInReal;
993
994       float lengthOfSecondLineOnPaper;
995       float lengthOfSecondLineInReal;
996
997       Polygon[] polygonsHavingThisPoint;
998
999       Point[] pointsOfPolygon;
1000      Point errorPoint;
1001      Polygon errorPolygon;
1002      Point pointOnLineInErrorPolygon0;
1003      Point pointOnLineInErrorPolygon1;
1004      Point counterpartPointOnCounterpartPolygon;
1005      Point pointOfOrigin;
1006
1007      Point pointOnFirstLine;
1008      Point pointOnSecondLine;
1009
1010      Vector[] possiableVector;
1011
1012      // get polygons having error point
1013      polygonsHavingThisPoint = getPolygons(changedPoint);
1014
1015      if(polygonsHavingThisPoint.length == 0) {
1016        int t=0;
1017      }
1018
1019      // check the polygons are fixable or not.
1020      for (i = 0; i < polygonsHavingThisPoint.length; i++) {
1021        pointsOfPolygon = polygonsHavingThisPoint[i].getPoints();
1022        errorPolygon = polygonsHavingThisPoint[i];
1023
1024        for (j = 0; j < Polygon.DEFAULT_NUMBER_OF_POINTS; j++) {
1025          if (pointsOfPolygon[j] == changedPoint) {
1026
1027            // get length of first line
1028            pointOnFirstLine = pointsOfPolygon[(j + 1) % 3];
1029
1030            lengthOfFirstLineOnPaper = getLine(pointsOfPolygon[j],
1031                pointOnFirstLine)
1032                .getLengthOnPaper();
1033
1034            lengthOfFirstLineInReal = getLine(pointsOfPolygon[j],
1035                pointOnFirstLine).getLengthInReal();
1036
1037            // get length of second line
1038            pointOnSecondLine = pointsOfPolygon[(j + 2) % 3];
1039
1040            lengthOfSecondLineOnPaper = getLine(pointsOfPolygon[j],
1041                pointOnSecondLine).getLengthOnPaper();
1042
1043            lengthOfSecondLineInReal = getLine(pointsOfPolygon[j],
1044                pointOnSecondLine).getLengthInReal();
1045
1046            if (lengthOfFirstLineInReal == Float.NaN
1047                || lengthOfSecondLineInReal == Float.NaN ) {
1048              int t=0;
1049
1050            }
1051
1052
1053            // two line error check
1054            if (!(Math.abs(lengthOfFirstLineInReal
1055                - lengthOfFirstLineOnPaper) <=
1056                 ERROR_RATIO_FOR_LANGTH)
1057              && !(Math.abs(lengthOfSecondLineInReal
1058                - lengthOfSecondLineOnPaper) <=
1059                  ERROR_RATIO_FOR_LANGTH)) {
1060              return false;
1061            }
1062
1063            if (lengthOfFirstLineInReal > 0.36
1064                || lengthOfSecondLineInReal > 0.36
1065               || lengthOfFirstLineInReal < 0
1066                || lengthOfSecondLineInReal < 0){
1067                  int q=0;
1068            }
1069          }
1070        }
1071
1072      }
1073
1074
1075      return true;
1076    }
1077
1078    boolean fixLinkage(Point changedPoint) {
1079      /*
1080       * error check
1081       *
```

```
     * 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;
        }
      }
    }
}
```

```
1218      // error 2:
1219      // inside line -> outside line
1220      // outside line -> inside line
1221      // 0-> inside
1222      // 0-> outside
1223  /**/
1224      for(i = 0; i<getNumberOfEdges(); i++ ){
1225        if (!(Math.abs(lines[i].getLengthInReal()- lines[i].getLengthOnPaper
              ()) < ERROR_RATIO_FOR_LANGTH)){
1226  /** /
1227          System.out.format("line length difference is biger than %f
                LengthInReal = %f LentghOnPaper = %f difference = %f\n"
1228              , ERROR_RATIO_FOR_LANGTH
1229              ,lines[i].getLengthInReal()
1230              ,lines[i].getLengthOnPaper()
1231              ,lines[i].getLengthInReal() - lines[i].getLengthOnPaper());
1232  /**/
1233          return false;
1234        }
1235        if (!(lines[i].getLengthInReal()< 3.7 )
1236            || !(lines[i].getLengthOnPaper() < 3.7 )) {
1237  /** /
1238          System.out.format("line length is biger than 3.7 LengthInReal = %f
                LentghOnPaper = %f\n"
1239              ,lines[i].getLengthInReal()
1240              ,lines[i].getLengthOnPaper());
1241  /**/
1242          return false;
1243        }
1244      }
1245  /**/
1246      return true;
1247    }
1248
1249    boolean fixAngleOnPolygons(Polygon polygon, Line line) {
1250      /*
1251       * return false : Way of angle is changed .; return true : way of Angle
1252       * not changed ;
1253       */
1254      Polygon[] polygonsSharedLine;
1255      Point point0;
1256      Point point1;
1257
1258      Vector vectorOfLineOnPolygon;
1259      Vector vectorFromClossProduct;
1260
1261      float oldAngle;
1262      float newAngle;
1263
1264      Vector unitVectorPolygon;
1265      Vector unitVectorCounterpartPolygon;
1266
1267      point0 = line.getStartPoint();
1268      point1 = line.getEndPoint();
1269
1270      unitVectorPolygon = polygon.getVectorFromLineToOppositePoint(point0,
1271          point1).getUnitVector();
1272      unitVectorCounterpartPolygon = getCounterpartPolygon(polygon, point0,
1273          point1).getVectorFromLineToOppositePoint(point0, point1)
1274          .getUnitVector();
1275
1276      unitVectorPolygon.invert();
1277      unitVectorPolygon = unitVectorPolygon.getUnitVector();
1278
1279      oldAngle = line.getAngle();
1280      newAngle = (float) Math.acos(Vector.dot(unitVectorPolygon,
1281          unitVectorCounterpartPolygon));
1282
1283      vectorOfLineOnPolygon = polygon.getVectorOnTheLine(point0, point1);
1284      vectorFromClossProduct = Vector.closs(unitVectorPolygon,
1285          unitVectorCounterpartPolygon);
1286
1287      // same way to line vector
1288      if (Vector.dot(unitVectorPolygon, unitVectorPolygon) > 0.99999999) {
1289
1290      } else {
1291        newAngle = -1 * newAngle;
1292      }
1293
1294      if (oldAngle * newAngle < 0) {
1295
1296        return false;
1297      } else {
1298
1299        return true;
1300      }
1301    }
1302
1303    // function for snapshot
1304    public Paper snapshot() {
1305      int i;
1306      Paper paper = new Paper();
1307
1308      Point[] points = new Point[numberOfPoints];
1309      for (i = 0; i < this.numberOfPoints; i++) {
1310        points[i] = this.pointsOnPaper[i].snapshot();
1311      }
1312
1313      Line[] lines = new Line[numberOfLines];
1314      for (i = 0; i < this.numberOfLines; i++) {
1315        lines[i] = this.lines[i].snapshot(paper, points);
1316      }
1317
1318      Polygon[] polygons = new Polygon[numberOfPolygons];
1319      for (i = 0; i < this.numberOfPolygons; i++) {
1320        polygons[i] = this.polygons[i].snapshot(paper, points);
1321      }
1322
1323      paper.setValue(points, numberOfPoints, lines, numberOfLines, polygons,
1324          numberOfPolygons);
1325
1326      return paper;
1327    }
1328
1329    public boolean changeAngleAsMuchAsPossible(Polygon polygon, Point point0,
1330        Point point1) {
1331      /**
1332       * Unfolding Edge as much as possible
1333       *
1334       */
1335
1336      int i;
1337
1338      boolean isAngleChanged;
1339
1340      float targetAngle;
1341      float stepAngle;
1342
1343      Line line;
1344      float foldingWay;
1345
1346      line = getLine(point0, point1);
1347
1348      if (line.getType() == Line.TYPE_POSITIVE_LINE) {
1349        foldingWay = 1.0f;
1350      } else if (line.getType() == Line.TYPE_NEGATIVE_LINE) {
1351        foldingWay = -1.0f;
1352      } else {
```

```
1353        return false;
1354      }
1355
1356      if (changeAngle(polygon, point0, point1, (float) 0)) {
1357        return true;
1358      }
1359
1360      stepAngle = (float) Math.PI / 2;
1361      targetAngle = (float) Math.PI / 2;
1362
1363      isAngleChanged = false;
1364      while (stepAngle != 0.0f) {
1365
1366        stepAngle /= 2;
1367
1368        if (changeAngle(polygon, point0, point1, foldingWay * targetAngle)) {
1369
1370          isAngleChanged = true;
1371          targetAngle -= stepAngle;
1372
1373        } else {
1374
1375          targetAngle += stepAngle;
1376        }
1377      }
1378
1379      return isAngleChanged;
1380    }
1381
1382    public boolean changeAngleForSmallestGlobalEnergy(Line line){
1383      boolean isAnglechanged = true;
1384
1385      return isAnglechanged;
1386    }
1387
1388    /**
1389     *
1390     * Unfolding Edge by target angles.
1391     *
1392     * 1. copy the paper. 2. pick a line. set the angle on line the target
1393     * point. 3. fix The Other Point around the point. 4. if fixing is false,
1394     * roll back. false state is there is more than 1 edge in one polygon we
1395     * have to fix. if fixing is success repeat 2 until there is no angle.
1396     *
1397     * Unfolding Edge by step
1398     */
1399
1400    public void setValue(Point[] points, int numberOfPoints, Line[] lines,
1401        int numberOfLines, Polygon[] polygons, int numberOfPolygons) {
1402
1403      this.pointsOnPaper = points;
1404      this.numberOfPoints = numberOfPoints;
1405      this.lines = lines;
1406      this.numberOfLines = numberOfLines;
1407      this.polygons = polygons;
1408      this.numberOfPolygons = numberOfPolygons;
1409
1410      buildSortedPoints();
1411
1412    }
1413
1414    public void setValue(Paper paper) {
1415
1416      setValue(paper.getPoints(), paper.getNumberOfPoints(),
1417          paper.getLines(), paper.getNumberOfEdges(),
1418          paper.getPolygons(), paper.getNumberOfPolygons());
1419
1420    }
1421
```

```
1422    Vector[] getVectorsOfPointsOnThreePoints(Point errorPoint, Point point0,
1423        Point point1, Point point2) {
1424      Vector v1, v2, v3;
1425      float length1, length2, length3;
1426
1427      v1 = point0.getVectorInReal();
1428      v2 = point1.getVectorInReal();
1429      v3 = point2.getVectorInReal();
1430
1431      length1 = getLine(point0, errorPoint).getLengthOnPaper();
1432      length2 = getLine(point1, errorPoint).getLengthOnPaper();
1433      length3 = getLine(point2, errorPoint).getLengthOnPaper();
1434
1435      return Vector.getVectorFrom3Vector(v1, length1, v2, length2, v3,
1436          length3);
1437    }
1438
1439    public float getGlobalEnergy() {
1440
1441      // energy = (float) Math.pow(energy,2) - 1.0f ;
1442
1443      // System.out.printf("Energy = %f\n " , energy);
1444      return getEnergy(pointsOnPaper);
1445    }
1446
1447    public void printAllOfLineLength() {
1448      int i;
1449      for (i = 0; i < numberOfLines; i++) {
1450        System.out.format("l in real =%f l on paper=%f \n", lines[i]
1451            .getLengthInReal(), lines[i].getLengthOnPaper());
1452      }
1453      System.out.format("\n");
1454    }
1455
1456    public float getLocalEnergy(Line line) {
1457
1458      int i, j;
1459
1460      ArrayList<Point> pointArray = new ArrayList<Point>();
1461
1462
1463      Point startPoint;
1464      Point endPoint;
1465
1466      pointArray.add(line.getStartPoint());
1467      pointArray.add(line.getEndPoint());
1468
1469      boolean isStartPointInPointArray = false;
1470      boolean isEndPointInPointArray = false;
1471
1472
1473      for (i=0; i <getNumberOfEdges(); i++) {
1474
1475        isStartPointInPointArray = false;
1476        isEndPointInPointArray = false;
1477
1478        startPoint = getLine(i).getStartPoint();
1479        endPoint = getLine(i).getEndPoint();
1480        for (j=0; j<pointArray.size(); j++) {
1481          if(startPoint == pointArray.get(j)) {
1482            isStartPointInPointArray = true;
1483          }
1484          if(endPoint == pointArray.get(j)) {
1485            isEndPointInPointArray = true;
1486          }
1487        }
1488
1489        if (!(isStartPointInPointArray & isEndPointInPointArray)
1490            && !isStartPointInPointArray){
```

```
1491            pointArray.add(startPoint);
1492          } else if (!(isStartPointInPointArray & isEndPointInPointArray)
1493            && !isEndPointInPointArray) {
1494            pointArray.add(endPoint);
1495          }
1496        }
1497
1498        return getEnergy((Point[]) pointArray.toArray());
1499      }
1500
1501      public static float getEnergy(Point[] points) {
1502        int i;
1503        int j;
1504
1505        float energy = 0.0f;
1506
1507        for (i = 0; i < points.length; i++) {
1508          for (j = 0; j < points.length; j++) {
1509            energy += (points[i].getXInReal() - points[j].getXInReal())
1510                * (points[i].getXInReal() - points[j].getXInReal())
1511                * (points[i].getXInReal() - points[j].getXInReal())
1512                + (points[i].getYInReal() - points[j].getYInReal())
1513                * (points[i].getYInReal() - points[j].getYInReal())
1514                * (points[i].getYInReal() - points[j].getYInReal())
1515                + (points[i].getZInReal() - points[j].getZInReal())
1516                * (points[i].getZInReal() - points[j].getZInReal())
1517                * (points[i].getZInReal() - points[j].getZInReal())
1518                * (points[i].getZInReal() - points[j].getZInReal());
1519
1520            energy -= (points[i].getXOnPaper() - points[j].getXOnPaper())
1521                * (points[i].getXOnPaper() - points[j].getXOnPaper())
1522                * (points[i].getXOnPaper() - points[j].getXOnPaper())
1523                + (points[i].getYOnPaper() - points[j].getYOnPaper())
1524                * (points[i].getYOnPaper() - points[j].getYOnPaper())
1525                * (points[i].getYOnPaper() - points[j].getYOnPaper())
1526                + (points[i].getZOnPaper() - points[j].getZOnPaper())
1527                * (points[i].getZOnPaper() - points[j].getZOnPaper())
1528                * (points[i].getZOnPaper() - points[j].getZOnPaper())
1529                * (points[i].getZOnPaper() - points[j].getZOnPaper());
1530          }
1531        }
1532
1533        energy = energy * energy;
1534
1535        return energy;
1536      }
1537    }
```

```
1  package com.drancom.programmableMatter.folding.controller.paper;
2
3  import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
4
5  public class Polygon {
6    public final static int DEFAULT_NUMBER_OF_POINTS = 3;
7
8    Paper paper;
9
10   Point points[];
11   int numberOfPoints;
12
13
14   public void setPolygon (Point[] points, Paper paper) throws Exception,
         NoLineException  {
15
16     int i;
17
18     int numberOfPoints;
19     int index_smallestCoordinatePoint;
20
21     Vector vector_smallestCoordinatePoint;
22     Vector vector;
23
24     this.paper = paper;
25
26     numberOfPoints = points.length;
27
28     for (i = 0; i < numberOfPoints; i++) {
29       if(paper.getLine(points[i], points [(i + 1) % numberOfPoints])== null
             ){
30         throw new NoLineException();
31       }
32     }
33
34     // find point having on the smallest  z < y < x ;
35     index_smallestCoordinatePoint = 0;
36     vector_smallestCoordinatePoint = points[index_smallestCoordinatePoint].
           getVectorOnPaper();
37
38     for (i = 0; i < numberOfPoints; i++) {
39       vector = points[i].getVectorOnPaper();
40       if(((vector_smallestCoordinatePoint.getZ() < vector.getZ())
41           || (vector_smallestCoordinatePoint.getZ() == vector.getZ() )
42               &&  vector_smallestCoordinatePoint.getY() < vector.getY())
43           || (vector_smallestCoordinatePoint.getZ() == vector.getZ()
44               &&  vector_smallestCoordinatePoint.getY() == vector.getY()
45               &&  vector_smallestCoordinatePoint.getX() < vector.getX()))
                   {
46         index_smallestCoordinatePoint=i;
47         vector_smallestCoordinatePoint = points[
               index_smallestCoordinatePoint].getVectorOnPaper();
48       }
49     }
50
51     this.points = new Point[numberOfPoints];
52
53     for(i = 0; i < numberOfPoints; i++) {
54       this.points[i] = points[(i+index_smallestCoordinatePoint) %
             numberOfPoints];
55       this.numberOfPoints = numberOfPoints;
56     }
57   }
58
59   public Point[] getPoints(){
60     return points;
61   }
62
63   public int getNumberOfPoints() {
64     return numberOfPoints;
65   }
66
67   public Point getOppositePoint(Point point0, Point point1) {
68     int i;
69     for(i=0; i<3; i++) {
70       if(points[i] != point0 && points[i] != point1)
71         return points[i];
72     }
73     return null;
74   }
75
76   public Vector getVectorOnTheLine(Point point0, Point point1) {
77     int i;
78     Vector vectorOnTheLine;
79     for(i=0 ; i < numberOfPoints; i++) {
80       if((points[i] == point0 && points[(i+1) % numberOfPoints] == point1)
```

```
81          || (points[i] == point1 && points[(i+1) % numberOfPoints] == point0
                ){
82          vectorOnTheLine = new Vector();
83          vectorOnTheLine.setXYZ(points[(i+1) % numberOfPoints].
                getVectorInReal());
84
85          vectorOnTheLine.subtractionVector(points[i].getVectorInReal());
86          return vectorOnTheLine;
87        }
88      }
89      return null;
90    }
91
92    public Vector getVectorFromLineToOppositePoint (Point point0, Point
          point1) {
93      int i;
94      Point oppositPoint;
95      Vector vectorFromLineToOppsitePoint;
96      Vector originVector;
97      if(null == paper.getLine(point0, point1)){
98        return null;
99      }
100
101     // vectorOnLineForStartPointOfVectorFromLineToOppsitePoint
102     oppositPoint = getOppositePoint(point0, point1);
103
104     originVector = getOriginVector(oppositPoint,
105         point0,
106         point1);
107
108
109     vectorFromLineToOppsitePoint = oppositPoint.getVectorInReal();
110
111     originVector.invert();
112     vectorFromLineToOppsitePoint.addVector(originVector);
113
114     return vectorFromLineToOppsitePoint;
115   }
116   public Vector getVectorOfStartPointFromLineToOppositePoint(Point point0,
          Point point1) {
117     Vector vectorFromStartPointToOppsitePoint;
118     Vector vectorOnLine;
119
120     Vector unitVectorAngle;
121
122     Vector vectorOnLineForStartPointOfVectorFromLineToOppsitePoint;
123
124
125     float angle;
126     float scaleOfVectorOnLineForStartPointOfVectorFromLineToOppsitePoint;
127
128
129     vectorOnLine = new Vector();
130
131     vectorFromStartPointToOppsitePoint = new Vector();
132
133
134     // end Point
135     vectorFromStartPointToOppsitePoint.setXYZ(point0.getVectorInReal());
136     // start point
137     vectorFromStartPointToOppsitePoint.subtractionVector(getOppositePoint(
            point0, point1).getVectorInReal());
138
139     // end Point
140     vectorOnLine.setXYZ(point0.getVectorInReal());
141     // start Point
142     vectorOnLine.subtractionVector(getOppositePoint(point0, point1).
            getVectorInReal());
143
144     // find angle
145     angle = (float) Math.acos((double) Vector.dot(
            vectorFromStartPointToOppsitePoint.getUnitVector(), vectorOnLine.
            getUnitVector()));
146
147     // find startPoint for Vector from line to opposite Point
148     scaleOfVectorOnLineForStartPointOfVectorFromLineToOppsitePoint = (
            float) ((float) vectorFromStartPointToOppsitePoint.getR() * Math.
            cos((float) angle));
149     vectorOnLineForStartPointOfVectorFromLineToOppsitePoint = vectorOnLine.
            getUnitVector();
150     vectorOnLineForStartPointOfVectorFromLineToOppsitePoint.scale(
            scaleOfVectorOnLineForStartPointOfVectorFromLineToOppsitePoint);
151
152     return vectorOnLineForStartPointOfVectorFromLineToOppsitePoint;
153   }
154   public boolean isAllPointsHave(Point point0, Point point1, Point point2)
          {
155     Point[] pointsAll = new Point[3];
156
157     pointsAll[0]  = point0;
158     pointsAll[1]  = point1;
159     pointsAll[2]  = point2;
160
161
162     return isAllPointsHave(pointsAll);
163   }
164
165   public boolean isAllPointsHave(Point[] points) {
166     int i;
167     int j;
168
169     int numberOfmachingPoints;
170     numberOfmachingPoints = 0;
171
172     for (i=0; i < points.length ; i++) {
173       for (j=0; j < this.points.length ; j++) {
174         if(this.points[j] == points[i]) {
175           numberOfmachingPoints++;
176           if(numberOfmachingPoints == points.length){
177             return true;
178           }
179         }
180       }
181     }
182     return false;
183   }
184
185   public boolean setAngle(Point point0, Point point1, float angle) throws
          NoLineException{
186     int i;
187     Line line;
188
189     for(i=0; i < numberOfPoints; i++) {
190       if ((points[i] == point0 && points[ ( i + 1 ) % numberOfPoints] ==
              point1)
191           || (points[i] == point1 && points[ ( i + 1) % numberOfPoints] ==
                point0)) {
192
193         return paper.changeAngle(this, point0, point1, angle);
194       }
195     }
196
197     throw new NoLineException();
198
199   }
200   public boolean isPointHaving(Point point) {
201     int i;
202
```

```
    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;

```

```
33    float levelOfActuratingPower = 0.0f;
34
35    Vector getVector() {
36      Vector vector = new Vector();
37      vector.setXYZ(endPoint.getXInReal() - startPoint.getXInReal()
38           , endPoint.getYInReal() - startPoint.getYInReal()
39           , endPoint.getZInReal() - startPoint.getZInReal());
40
41      return vector;
42    }
43    public int getType(){
44      return type;
45    }
46
47    public Point getStartPoint(){
48      return startPoint;
49    }
50
51    public Point getEndPoint(){
52      return endPoint;
53    }
54
55    public float getAngle(){
56      int i;
57  /** /
58      float angleBetweenVectors;
59      float angle;
60      Polygon[] polygons;
61      Vector v1, v2;
62
63      if (type == TYPE_EDGE_LINE) {
64        return Float.intBitsToFloat(0x7fc00000); //Not a number
65      }
66
67      polygons = paper.getPolygons(startPoint, endPoint);
68      if (polygons.length != 2) {
69        return Float.intBitsToFloat(0x7fc00000); //Not a number
70      }
71
72      v1= polygons[0].getVectorFromLineToOppositePoint(startPoint, endPoint);
73      v2= polygons[1].getVectorFromLineToOppositePoint(startPoint, endPoint);
74      angleBetweenVectors = (float) Math.acos(Vector.dot(v1, v2));
75
76      if (type == TYPE_POSITIVE_LINE){
77        angle = (float) Math.PI - angleBetweenVectors;
78      } else  if (type == TYPE_NEGATIVE_LINE){
79        angle = -1 * ( (float) Math.PI -  angleBetweenVectors);
80      } else {
81        angle = (float) Math.PI - angleBetweenVectors;
82      }
83  /**/
84  //    this.angle = angle;
85
86      return angle;
87
88    }
89
90    public float getLengthOnPaper() {
91      Vector v1, v2;
92      v1 = startPoint.getVectorOnPaper();
93      v2 = endPoint.getVectorOnPaper();
94      v2.invert();
95      v1.addVector(v2);
96
97      return  v1.getR();
98    }
99
100   public float getLengthInReal() {
101     Vector v1, v2;
102     v1 = startPoint.getVectorInReal();
103     v2 = endPoint.getVectorInReal();
104     v2.invert();
105     v1.addVector(v2);
106
107     return  v1.getR();
108   }
109
110   public void setLevelOfActuratingPower(float levelOfActuratingPower){
111     this.levelOfActuratingPower = levelOfActuratingPower;
112   }
113
114   public float getLevelOfActuratingPower(){
115     return levelOfActuratingPower;
116   }
117
118   public boolean isActurating(){
119
120     if (levelOfActuratingPower==0.0) {
121
122       return false;
123
124     } else {
125       return true;
126     }
127   }
128
129
130   public void setAngle(float angle) {
131
132     this.angle = angle;
133
134     if(this.angle == 0.0) {
135       this.type = TYPE_STATIC_LINE;
136     } else if(angle > 0.0) {
137       this.type = TYPE_POSITIVE_LINE;
138     } else {
139       this.type = TYPE_NEGATIVE_LINE;
140     }
141   }
142
143   public void addAngle(float angle) {
144
145     this.angle += angle;
146
147     if(this.angle == 0.0) {
148       this.type = TYPE_STATIC_LINE;
149     } else if(angle > 0.0) {
150       this.type = TYPE_POSITIVE_LINE;
151     } else {
152       this.type = TYPE_NEGATIVE_LINE;
153     }
154   }
155
156   public void setLine(Paper paper,
157             Point startPoint,
158             Point endPoint,
159             float angle,
160             int typeOfLine){
161
162     this.paper = paper;
163
164
165     if (startPoint.getXOnPaper() == endPoint.getXOnPaper()) {
166       if (startPoint.getYOnPaper() < endPoint.getYOnPaper()) {
167
168         this.startPoint = startPoint;
169         this.endPoint = endPoint;
170
```

```
171         } else {
172
173           this.startPoint = endPoint;
174           this.endPoint = startPoint;
175
176         }
177       } else if (startPoint.getYOnPaper() == endPoint.getYOnPaper()) {
178         if (startPoint.getXOnPaper() < endPoint.getXOnPaper()) {
179
180           this.startPoint = startPoint;
181           this.endPoint = endPoint;
182
183         } else {
184           this.startPoint = endPoint;
185           this.endPoint = startPoint;
186         }
187
188       } else if (startPoint.getXOnPaper() < endPoint.getXOnPaper()) {
189         this.startPoint = startPoint;
190         this.endPoint   = endPoint;
191       } else {
192         this.startPoint = endPoint;
193         this.endPoint   = startPoint;
194       }
195
196
197       this.angle = angle;
198       this.type = typeOfLine;
199
200       if(startPoint.getXOnPaper() == endPoint.getXOnPaper()
201           || startPoint.getYOnPaper() == endPoint.getYOnPaper()){
202
203         this.directionOfLine = DIRECTION_STRIGHT_LINE;
204
205       } else{
206         this.directionOfLine = DIRECTION_OBLIQUE_LINE;
207         startPoint.setType(Point.TYPE_OBLIQUE_LINE_CLOSS);
208         endPoint.setType(Point.TYPE_OBLIQUE_LINE_CLOSS);
209       }
210
211
212
213       if (typeOfLine == TYPE_STATIC_LINE){
214         maxPositiveAngle = 0;
215         maxNegativeAngle = 0;
216       }
217
218       if (typeOfLine == TYPE_POSITIVE_LINE){
219         maxPositiveAngle = 180;
220         maxNegativeAngle = 0;
221       }
222
223       if (typeOfLine == TYPE_NEGATIVE_LINE){
224         maxPositiveAngle = 0;
225         maxNegativeAngle = 180;
226       }
227
228       if (typeOfLine == TYPE_BOTHWAY_LINE){
229         maxPositiveAngle = 180;
230         maxNegativeAngle = 180;
231       }
232     }
233
234     public Line snapshot(Paper paper, Point [] newSetOfPoints) {
235       int i;
236
237       Line line = new Line();
238       Point startPoint = null;
239       Point endPoint = null;
```

```
240
241
242       for(i=0; i<newSetOfPoints.length; i++){
243         if(newSetOfPoints[i].getXOnPaper() == this.startPoint.getXOnPaper()
244             && newSetOfPoints[i].getYOnPaper() == this.startPoint.getYOnPaper
                    ()
245             && newSetOfPoints[i].getZOnPaper() == this.startPoint.getZOnPaper
                    ()){
246           startPoint = newSetOfPoints[i];
247         } else if(newSetOfPoints[i].getXOnPaper() == this.endPoint.
                getXOnPaper()
248             && newSetOfPoints[i].getYOnPaper() == this.endPoint.getYOnPaper
                    ()
249             && newSetOfPoints[i].getZOnPaper() == this.endPoint.getZOnPaper
                    ()){
250           endPoint = newSetOfPoints[i];
251         }
252
253         if(startPoint != null && endPoint != null ){
254           line.setValues(paper, maxPositiveAngle, maxNegativeAngle,
                  startPoint, endPoint, angle, type, directionOfLine);
255           return line;
256         }
257       }
258
259       return null;
260     }
261
262     void setValues(
263         Paper paper
264         ,float maxPositiveAngle
265         ,float maxNegativeAngle
266
267         , Point startPoint
268         , Point endPoint
269
270         ,float angle
271
272         , int type
273         , int directionOfLine) {
274
275       this.paper = paper;
276
277       this.maxPositiveAngle = maxPositiveAngle;
278       this.maxNegativeAngle = maxNegativeAngle;
279
280       this.startPoint = startPoint;
281       this.endPoint   = endPoint;
282
283       this.angle = angle;
284
285       this.type = type;
286       this.directionOfLine = directionOfLine;
287
288     }
289   }
```

```
1   package com.drancom.programmableMatter.folding.controller.paper;
2
3   import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
4
5   public class Point {
6
7     public final static int PLACE_OF_DECIMAL_POINT = 4;
8
9     public final static int TYPE_ONLY_STRIGHT_LINE_CLOSS  = 0;
10    public final static int TYPE_OBLIQUE_LINE_CLOSS     = 1;
11
12    boolean isRenewed;
```

```
13
14    Vector vOnPaper;
15
16    Vector vInReal;
17
18
19    private int type;
20
21    public Point () {
22      isRenewed = false;
23      vOnPaper = new Vector();
24      vInReal  = new Vector();
25
26      type = TYPE_ONLY_STRIGHT_LINE_CLOSS;
27    }
28
29    public float getXOnPaper() {
30      return vOnPaper.getX();
31    }
32
33    public float getYOnPaper() {
34      return vOnPaper.getY();
35    }
36
37    public float getZOnPaper() {
38      return vOnPaper.getZ();
39    }
40
41    public float getXInReal() {
42      return vInReal.getX();
43    }
44
45    public float getYInReal() {
46      return vInReal.getY();
47    }
48
49    public float getZInReal() {
50      return vInReal.getZ();
51    }
52
53    public void setType(int type){
54      this.type = type;
55    }
56
57    public int getType(){
58      return type;
59    }
60
61
62    public Vector getVectorInReal (){
63      Vector vector = new Vector();
64
65      vector.setXYZ(vInReal);
66
67      return vector;
68    }
69
70    public Vector getVectorOnPaper (){
71      Vector vector = new Vector();
72
73      vector.setXYZ(vOnPaper);
74
75      return vector;
76    }
77
78    public void setRenewed(boolean isRenewed) {
79      this.isRenewed = isRenewed;
80    }
81
```

```
82
83    public boolean isRenewed(){
84      return isRenewed;
85    }
86
87    public void setPointOnPaper(float xOnPaper, float yOnPaper, float
          zOnPaper) {
88      vOnPaper.setXYZ(xOnPaper,yOnPaper,zOnPaper);
89    }
90
91    public void setPointInReal (float xInReal,  float yInReal,  float zInReal
          ) {
92      vInReal.setXYZ(xInReal, yInReal, zInReal);
93    }
94
95    public void setVectorInReal (Vector vector){
96      setPointInReal(vector.getX(), vector.getY(), vector.getZ());
97    }
98
99    public void setVectorOnPaper (Vector vector){
100     setPointOnPaper(vector.getX(), vector.getY(), vector.getZ());
101   }
102
103   public Point snapshot(){
104     Point point = new Point();
105     point.setValues(isRenewed, vOnPaper, vInReal, type);
106
107     return point;
108   }
109
110   void setValues( boolean isRenewed
111       , Vector vOnPaper
112       , Vector vInReal
113       , int type  ){
114     this.isRenewed = isRenewed;
115
116     this.vOnPaper.setXYZ(vOnPaper);
117     this.vInReal.setXYZ(vInReal);
118
119     this.type = type;
120
121   }
122 }
```

```
1  package com.drancom.programmableMatter.folding.controller.paper;
2
3  public class NoLineException extends Exception {
4
5  }
```

```
1  package com.drancom.programmableMatter.folding.controller.paper;
2
3  public class EnergyFunction {
4
5    public static float getGlobalEnergy(Paper paper) {
6      float energy = 0.0f;
7      int i, j;
8
9      for(i = 0 ; i<paper.numberOfPoints ; i++) {
10       for(j = i + 1; j < paper.numberOfPoints ; j++) {
11         energy += (paper.pointsOnPaper[i].getXInReal()  - paper.
               pointsOnPaper[j].getXInReal()) * (paper.pointsOnPaper[i].
               getXInReal() - paper.pointsOnPaper[j].getXInReal())
12           + (paper.pointsOnPaper[i].getYInReal()   - paper.pointsOnPaper[j
                 ].getYInReal()) * (paper.pointsOnPaper[i].getYInReal() -
                 paper.pointsOnPaper[j].getYInReal())
13           + (paper.pointsOnPaper[i].getZInReal()   - paper.pointsOnPaper[j
                 ].getZInReal()) * (paper.pointsOnPaper[i].getZInReal() -
                 paper.pointsOnPaper[j].getZInReal());
```

```
14          energy -= (paper.pointsOnPaper[i].getXOnPaper() - paper.
                pointsOnPaper[j].getXOnPaper())   * (paper.pointsOnPaper[i].
                getXOnPaper() - paper.pointsOnPaper[j].getXOnPaper())
15            +  (paper.pointsOnPaper[i].getYOnPaper()     - paper.pointsOnPaper
                  [j].getYOnPaper())   * (paper.pointsOnPaper[i].getYOnPaper()
                  - paper.pointsOnPaper[j].getYOnPaper())
16            +  (paper.pointsOnPaper[i].getZOnPaper()     - paper.pointsOnPaper
                  [j].getZOnPaper())   * (paper.pointsOnPaper[i].getZOnPaper()
                  - paper.pointsOnPaper[j].getZOnPaper());
17        }
18      }
19      energy = energy * energy;
20  //     energy = (float) Math.exp(energy) - 1.0f ;
21
22      System.out.printf("Energy = %f" , energy);
23      return energy;
24    }
25  }
```

```
1  package com.drancom.programmableMatter.folding.controller.paper;
2
3  public class UnfoldingPaper extends Paper {
4    public boolean unfoldingEdge(int EdgeId, int polygonId, float stepAngle)
         {
5      Paper paper = this.snapshot();
6
7
8
9      return true;
10   }
11
12
13
14
15
16   public UnfoldingPaper snapshot() {
17     int i;
18     UnfoldingPaper paper = new UnfoldingPaper ();
19
20     Point[] points = new Point [numberOfPoints];
21     for(i=0; i<this.numberOfPoints; i++) {
22       points[i] = this.pointsOnPaper[i].snapshot();
23     }
24
25     Line[] lines = new Line [numberOfLines];
26     for(i=0; i<this.numberOfLines; i++) {
27       lines[i] = this.lines[i].snapshot(paper, points);
28     }
29
30     Polygon[] polygons = new Polygon[numberOfPolygons];
31     for(i=0; i<this.numberOfPolygons; i++) {
32       polygons[i] = this.polygons[i].snapshot(paper, points);
33     }
34
35     paper.setValue(points, numberOfPoints, lines, numberOfLines,polygons,
           numberOfPolygons);
36
37     return paper;
38
39   }
40
41 }
```

```
1  package com.drancom.programmableMatter.folding.controller.paper.util;
2
3  import com.wolfram.jlink.KernelLink;
4  import com.wolfram.jlink.MathLinkException;
5  import com.wolfram.jlink.MathLinkFactory;
6
7  public class Mathematica {
8    static KernelLink ml = null;
9    static int users = 0;
10
11   public KernelLink load() {
12     try {
13       ml = MathLinkFactory.createKernelLink("-linkmode launch -linkname 'c
               :\\program files\\wolfram research\\mathematica\\6.0\\mathkernel
               .exe'");
14       // Get rid of the initial InputNamePacket the kernel will send
15       // when it is launched.
16       ml.discardAnswer();
17     } catch (MathLinkException e) {
18       System.out.println("Fatal error opening link: " + e.getMessage());
19       return null;
20     }
21
22
23     return ml;
24   }
25   public KernelLink getKernelLink(){
26
27     return ml;
28   }
29
30   public void close() {
31     users--;
32
33   }
34
35
36
37
38 }
```

```
1  package com.drancom.programmableMatter.folding.controller.paper.util;
2
3  import com.drancom.programmableMatter.folding.controller.paper.Point;
4  import com.wolfram.jlink.Expr;
5  import com.wolfram.jlink.KernelLink;
6  import com.wolfram.jlink.MathLinkException;
7  import com.wolfram.jlink.MathLinkFactory;
8
9  import quicktime.qd3d.math.Vector3D;
10 import sun.awt.GlobalCursorManager;
11 import sun.reflect.ReflectionFactory.GetReflectionFactoryAction;
12
13 public class Vector {
14
15 //   final static int PLACE_OF_DECIMALS = 10;
16   final static float NUMBER_FOR_ROUNDING = 0.5f;
17
18   private float tenPowOfPLACE_OF_DECIMALS; // ten Power Of
         PLACE_OF_DECIMALS
19
20   private float x;
21   private float y;
22   private float z;
23
24   private float r;
25   private float theta;
26   private float phi;
27
28   public Vector() {
29 //     tenPowOfPLACE_OF_DECIMALS = Math.pow(10, PLACE_OF_DECIMALS);
30   }
31
32   public float getX(){
```

```
33      return x;
34    }
35    public float getY(){
36      return y;
37    }
38    public float getZ(){
39      return z;
40    }
41
42    public float getR(){
43      return r;
44    }
45    public float getTheta(){
46      return theta;
47    }
48    public float getPhi(){
49      return phi;
50    }
51
52    public float getLength(){
53
54
55      return r;
56    }
57
58    public void setXYZ(Vector vector){
59      setXYZ(vector.getX(), vector.getY(), vector.getZ());
60    }
61
62    public void setXYZ(float[] vectorPoints){
63      setXYZ(vectorPoints[0], vectorPoints[1], vectorPoints[2]);
64    }
65
66    public void setXYZ(float x, float y, float z){
67
68      float[]  vectorTransformed;
69
70  /**/
71      this.x = x;
72      this.y = y;
73      this.z = z;
74  /** /
75      this.x  = ((float)(int)(( x * tenPowOfPLACE_OF_DECIMALS)+
              NUMBER_FOR_ROUNDING))/ tenPowOfPLACE_OF_DECIMALS;
76      this.y  = ((float)(int)(( y * tenPowOfPLACE_OF_DECIMALS)+
              NUMBER_FOR_ROUNDING))/ tenPowOfPLACE_OF_DECIMALS;
77      this.z  = ((float)(int)(( z * tenPowOfPLACE_OF_DECIMALS)+
              NUMBER_FOR_ROUNDING))/ tenPowOfPLACE_OF_DECIMALS;
78  /**/
79      vectorTransformed = transformXYZtoRThetaPhi(x, y, z);
80  /**/
81      r     = vectorTransformed[0];
82      theta   = vectorTransformed[1];
83      phi   = vectorTransformed[2];
84  /** /
85      r     = ((float)(int)(( vectorTransformed[0] *
              tenPowOfPLACE_OF_DECIMALS)+NUMBER_FOR_ROUNDING))/
              tenPowOfPLACE_OF_DECIMALS;
86      theta   = ((float)(int)(( vectorTransformed[1] *
              tenPowOfPLACE_OF_DECIMALS)+NUMBER_FOR_ROUNDING))/
              tenPowOfPLACE_OF_DECIMALS;
87      phi   = ((float)(int)(( vectorTransformed[2] *
              tenPowOfPLACE_OF_DECIMALS)+NUMBER_FOR_ROUNDING))/
              tenPowOfPLACE_OF_DECIMALS;
88  /**/
89    }
90
91    public void setRThetaPhi(float[] vector){
92      setRThetaPhi(vector[0], vector[1], vector[2]);
93    }
94
95    public void setRThetaPhi(float r, float theta, float phi){
96
97      float[]  vectorTransformed;
98  /**/
99      this.r     = r     ;
100     this.theta = theta ;
101     this.phi   = phi   ;
102 /** /
103     this.r     = ((float)(int)(( r     * tenPowOfPLACE_OF_DECIMALS)+
              NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
104     this.theta = ((float)(int)(( theta * tenPowOfPLACE_OF_DECIMALS)+
              NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
105     this.phi   = ((float)(int)(( phi   * tenPowOfPLACE_OF_DECIMALS)+
              NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
106
107 /**/
108     vectorTransformed = transformRThetaPhitoXYZ(r, theta, phi);
109 /**/
110     this.x = vectorTransformed[0];
111     y = vectorTransformed[1];
112     z = vectorTransformed[2];
113 /** /
114     x = ((float)(int)(( vectorTransformed[0] * tenPowOfPLACE_OF_DECIMALS)+
              NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
115     y = ((float)(int)(( vectorTransformed[1] * tenPowOfPLACE_OF_DECIMALS)+
              NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
116     z = ((float)(int)(( vectorTransformed[2] * tenPowOfPLACE_OF_DECIMALS)+
              NUMBER_FOR_ROUNDING)) / tenPowOfPLACE_OF_DECIMALS;
117 /**/
118   }
119   public static double asin(double a) {
120     if (a>1) {
121       a=1;
122     } else if (a<-1) {
123       a=-1;
124     }
125
126     return Math.asin(a);
127   }
128   public static double acos(double a) {
129     if (a>1) {
130       a=1;
131     } else if (a<-1) {
132       a=-1;
133     }
134
135
136     return Math.acos(a);
137   }
138
139
140   public static float[] transformXYZtoRThetaPhi(float x, float y, float z){
141
142     float[]  vectorTransformed;
143
144     vectorTransformed = new float[3]; // r Theta Phi
145     float r, theta, phi;
146     float a;
147
148     r = (float) Math.sqrt( ((float) x * (float) x )+ ((float) y * (float) y
              ) + ((float) z * (float) z));  // r
149     phi =0;
150     theta=0;
151     a = 0;
152
153     if( r == 0) {
154       phi =0;
```

```
155         theta =0;
156
157       } else {
158
159         phi  = (float) acos( (double) (z / r) );// Phi   r cos Phi = z   acos
                z/r
160
161         a = (float) (x / (r * Math.sin((float) phi)));
162         if (a>1.0){
163           a=1;
164         } else if (a<-1.0) {
165           a=-1;
166         }
167         theta = (float) acos(a);  // Theta r sin phi * cos theta = x
168                                                                  // theta = acos
                                                                       (( x / r ) /
                                                                        sin phi )
169   //      vectorTransformed[1] = (float) Math.atan((float) (y / x));
             // Theta tan Theta = y / x
170       }
171
172       vectorTransformed[0] = (float) r;
173       vectorTransformed[1] = (float) theta;
174       vectorTransformed[2] = (float) phi;
175
176       return vectorTransformed;
177   }
178
179   public static float[] transformRThetaPhitoXYZ(float r, float theta, float
          phi){
180     float[]  vectorTransformed;
181     vectorTransformed = new float[3];
182     if (r==0) {
183       vectorTransformed[0] = 0.0f;  // x
184       vectorTransformed[1] = 0.0f;  // y
185       vectorTransformed[2] = 0.0f;                  // z
186     } else {
187       vectorTransformed[0] = (float)( r * Math.sin((float) phi) * Math.cos
            ((float) theta));   // x
188       vectorTransformed[1] = (float)( r * Math.sin((float) phi) * Math.sin
            ((float) theta));   // y
189       vectorTransformed[2] = (float)( r * Math.cos((float) phi));
                // z
190     }
191     return vectorTransformed;
192   }
193   public Vector getUnitVector(){
194     Vector unitVector = new Vector();
195     float x, y, z;
196     x = this.x / this.r;
197     y = this.y / this.r;
198     z = this.z / this.r;
199
200     unitVector.setXYZ(x, y, z);
201     return unitVector;
202   }
203
204   public void addVector(Vector startPointOfShardlineVector) {
205     transform(this, startPointOfShardlineVector);
206   }
207
208   public void subtractionVector(Vector vector) {
209     Vector invertVector = new Vector();
210
211     invertVector.setXYZ(vector.getX(), vector.getY(), vector.getZ());
212     invertVector.invert();
213
214     transform(this, invertVector);
215   }
```

```
216
217   public void invert() {
218     invert(this);
219   }
220
221
222
223   public void transform(Vector transformVector) {
224     transform(this, transformVector);
225   }
226   public void scale(float scale){
227     scale(this, scale);
228   }
229
230   public void scale(Vector scaleVector){
231     scale(this, scaleVector);
232   }
233
234   public float dot(Vector vector){
235     return dot(this, vector);
236   }
237
238   public void closs(Vector vector) {
239     closs(this, vector);
240   }
241
242
243   public void rotationX(float angle){
244     rotationX(this, angle);
245   }
246
247   public void rotationY(float angle){
248     rotationY(this, angle);
249
250   }
251   public void rotationZ(float angle){
252     rotationZ(this, angle);
253   }
254   public void rotation(Vector axisVector, float angle){
255     rotation(this, axisVector, angle);
256   }
257
258   public static void invert(Vector vector) {
259     vector.setXYZ(-1 * vector.x, -1 * vector.y, -1 * vector.z);
260
261   }
262
263   public static void transform(Vector vector, Vector transformVector){
264     float x, y, z;
265
266     x= vector.getX() + transformVector.getX();
267     y= vector.getY() + transformVector.getY();
268     z= vector.getZ() + transformVector.getZ();
269
270     vector.setXYZ(x,y,z);
271   }
272
273   public static void rotationX(Vector vector, float angle){
274     float x, y, z;
275   /**/
276     x = (float) ( 1                 *  vector.x
277             + 0                       *  vector.y
278             + 0                  *  vector.z);
279
280     y = (float) ( 0                    *  vector.x
281             +      Math.cos((double) angle) *  vector.y
282             + -1 * Math.sin((double) angle) *  vector.z);
283
284     z = (float) ( 0                    *  vector.x
```

```
285             + Math.sin((double) angle)     *  vector.y
286             + Math.cos((double) angle)     *  vector.z);
287     vector.setXYZ(x, y, z);
288
289   }
290
291   public static void rotationY(Vector vector, float angle){
292     float x, y, z;
293
294     x = (float) (      Math.cos((double) angle)     *  vector.x
295             + 0                            *  vector.y
296             + Math.sin((double) angle)     *  vector.z);
297
298     y = (float) ( 0                    *  vector.x
299             + 1                             *  vector.y
300             + 0                  *  vector.z);
301
302     z = (float) (-1 * Math.sin((double) angle)     *  vector.x
303             + 0                             *  vector.y
304             + Math.cos((double) angle)     *  vector.z);
305
306     vector.setXYZ(x, y, z);
307
308   }
309   public static void rotationZ(Vector vector, float angle){
310     float x, y, z;
311
312     x = (float) ((double) Math.cos((double)      angle)     *  vector.x
313             + -1 * (double) Math.sin((double) angle) *  vector.y
314             + 0                            *  vector.z);
315
316     y = (float) ((double) Math.sin((double) angle      *  vector.x
317             + (double) Math.cos((double) angle))     *  vector.y
318             + 0                          *  vector.z);
319
320     z = (float) ( 0                     *  vector.x
321             + 0                               *  vector.y
322             + 1                          *  vector.z);
323
324     vector.setXYZ(x, y, z);
325   }
326
327   public static void rotation(Vector vector, Vector axisVector, float angle
        ){
328     /**
329      *  1 + (1-cos(angle))*(x*x-1) -z*sin(angle)+(1-cos(angle))*x*y y*sin(
           angle)+(1-cos(angle))*x*z
330      *  z*sin(angle)+(1-cos(angle))*x*y 1 + (1-cos(angle))*(y*y-1) -x*sin(
           angle)+(1-cos(angle))*y*z
331      * -y*sin(angle)+(1-cos(angle))*x*z x*sin(angle)+(1-cos(angle))*y*z 1 +
            (1-cos(angle))*(z*z-1)
332      */
333
334     float x, y, z;
335     float rOnAxis, thetaOnAxis, phiOnAxis;
336     float r, theta, pi;
337     Vector unitVector;
338
339     unitVector = axisVector.getUnitVector();
340
341
342 /**/ // rotation func 1
343     x = (float) ((( 1 + (1 - Math.cos(angle)) * (unitVector.x * unitVector.
            x - 1))                          * vector.x)
344         +    (( -1 * unitVector.z * Math.sin(angle) + (1 - Math.cos(angle))
                * unitVector.x * unitVector.y)  * vector.y)
345         +    ((      unitVector.y * Math.sin(angle) + (1 - Math.cos(angle))
                * unitVector.x * unitVector.z)  * vector.z));
346
347     y = (float) ((( unitVector.z * Math.sin(angle) + (1 - Math.cos(angle))
            * unitVector.x * unitVector.y)      * vector.x)
348         +  (( 1 + (1 - Math.cos(angle)) * (unitVector.y * unitVector.y - 1)
                )                          * vector.y)
349         +  ((-1 * unitVector.x * Math.sin(angle) + (1 - Math.cos(angle)) *
                unitVector.y * unitVector.z) * vector.z));
350
351     z = (float) (((-1 * unitVector.y * Math.sin(angle) + (1 - Math.cos(
            angle)) * unitVector.x * unitVector.z) * vector.x)
352         +  ((    unitVector.x * Math.sin(angle) + (1 - Math.cos(angle)) *
                unitVector.y * unitVector.z) * vector.y)
353         +     (( 1 + (1 - Math.cos(angle)) * (unitVector.z * unitVector.z -
                1))                          * vector.z));
354     vector.setXYZ(x, y, z);
355
356 /** /  // rotation func 2
357     rOnAxis = axisVector.getR();
358     thetaOnAxis = axisVector.getTheta();
359     phiOnAxis = axisVector.getPhi();
360
361     r = vector.getR();
362     theta = vector.getTheta();
363     pi = vector.getPhi();
364
365
366     rotationZ(vector,  1 * thetaOnAxis);
367     rotationY(vector,  1 * phiOnAxis);
368
369     rotationX(vector, angle);
370
371     rotationY(vector, -1 * phiOnAxis);
372     rotationZ(vector, -1 * thetaOnAxis);
373
374 /**/
375   }
376
377   public static float dot(Vector vector0, Vector vector1){
378     return vector0.getX() * vector1.getX()
379       + vector0.getY() * vector1.getY()
380       + vector0.getZ() * vector1.getZ();
381   }
382
383   public static Vector closs (Vector vector0, Vector vector1) {
384     float x0,y0,z0;
385     float x1,y1,z1;
386
387     Vector vector = new Vector();
388
389     x0 = vector0.getX();
390     y0 = vector0.getY();
391     z0 = vector0.getZ();
392
393     x1 = vector1.getX();
394     y1 = vector1.getY();
395     z1 = vector1.getZ();
396
397     vector.setXYZ((y0*z1 - z0*y1), -1 * (x0 * z1 - z0 * x1 ),  (x0 * y1 -
            y0 * x1));
398
399     return vector;
400   }
401
402
403   public static void scale(Vector vector, float scale){
404     vector.setXYZ(scale * vector.getX()
405         , scale * vector.getY()
406         , scale * vector.getZ());
407     }
408
```

```
409   public static void scale(Vector vector, Vector scaleVector){
410     scale(vector, scaleVector.getR());
411     }
412
413
414
415   public static Vector [] getVectorFrom3Vector(Vector v1, float l1,  //
          length between point1 and point4
416       Vector v2, float l2, // length between point2 and point4
417       Vector v3, float l3){ // length between point3 and point4
418
419     int i;
420     int j;
421
422     // Initiation vectorTemp
423     Vector [] vectorTemp = new Vector[3];
424     for (i=0 ; i <3; i++) {
425       vectorTemp[i] = new Vector();
426     }
427
428     Vector vForTransforming = new Vector();
429
430
431     // Initiation v4
432     Vector [] v4 = new Vector [2];
433     for (i=0 ; i <2; i++) {
434       v4[i] = new Vector();
435     }
436
437     vectorTemp[0].setXYZ(v1);
438     vectorTemp[1].setXYZ(v2);
439     vectorTemp[2].setXYZ(v3);
440
441     float a1, b1, c1;
442     float a2, b2, c2;
443     float a3, b3, c3;
444
445     a1 = vectorTemp[0].getX();
446     b1 = vectorTemp[0].getY();
447     c1 = vectorTemp[0].getZ();
448
449     a2 = vectorTemp[1].getX();
450     b2 = vectorTemp[1].getY();
451     c2 = vectorTemp[1].getZ();
452
453     a3 = vectorTemp[2].getX();
454     b3 = vectorTemp[2].getY();
455     c3 = vectorTemp[2].getZ();
456
457
458     String EQ1 = String.format("EQ1:=(x- (%15f))^2+(y- (%15f))^2+(z- (%15f)
          )^2- %15f^2==0",a1,b1,c1,l1 );
459     String EQ2 = String.format("EQ2:=(x- (%15f))^2+(y- (%15f))^2+(z- (%15f)
          )^2- %15f^2==0",a2,b2,c2,l2 );
460     String EQ3 = String.format("EQ3:=(x- (%15f))^2+(y- (%15f))^2+(z- (%15f)
          )^2- %15f^2==0",a3,b3,c3,l3 );
461     String Solve = String.format("NSolve[{EQ1,EQ2,EQ3},{x,y,z}]");
462
463     Mathematica mathematica= new Mathematica();
464
465     KernelLink ml = mathematica.getKernelLink();
466
467     try {
468
469       ml.evaluate(EQ1);
470       ml.discardAnswer();
471       ml.evaluate(EQ2);
472       ml.discardAnswer();
473       ml.evaluate(EQ3);
474       ml.discardAnswer();
475
476       String result ;
477       result = ml.evaluateToOutputForm(Solve, 0);
478 //      System.out.println(result);
479
480       setVectorFromStringOfResultFromMathmatica(v4, result);
481
482
483     } catch (MathLinkException e) {
484       System.out.println("MathLinkException occurred: " + e.getMessage());
485     } finally {
486     }
487
488     return v4;
489   }
490
491
492   private static boolean setVectorFromStringOfResultFromMathmatica(Vector[]
          v4,
493       String result) {
494
495     float [] x = new float [2];
496     float [] y = new float [2];
497     float [] z = new float [2];
498     int exponentialNumber[] = new int[6] ;
499
500
501     int indexOfX=0;
502     int indexOfY=0;
503     int indexOfZ=0;
504
505     int indexOfExponentialNumber = 0;
506     int numberOfExponentialNumber = 0;
507
508
509     int i;
510
511
512     String tokenFirstSplit[];
513     String tokenFirstLine[];
514     String tokenSecondLine[];
515
516         //read command
517
518         result = result.replace('}', ' ');
519         result = result.replace('{', ' ');
520         result = result.replaceAll("  ", " ");
521         result = result.replaceAll(",", " ");
522         result = result.trim();
523
524         try {
525
526           tokenFirstSplit = result.split("\n");
527
528         } catch (NullPointerException e) {
529
530           return false;
531         }
532
533         if (tokenFirstSplit.length == 2) {
534           try {
535             tokenFirstLine = tokenFirstSplit[0].split(" ");
536             tokenSecondLine = tokenFirstSplit[1].split(" ");
537             } catch (NullPointerException e) {
538               return false;
539             }
540
541           for (i=0; i<tokenFirstLine.length; i++) {
```

```
542
543             if (tokenFirstLine[i].equals("")) {
544
545             } else if (0 != Integer.parseInt(tokenFirstLine[i])) {
546               if(indexOfExponentialNumber >= 5){
547                 int t=0;
548               }
549                 exponentialNumber[indexOfExponentialNumber]
550                                   = Integer.parseInt(tokenFirstLine[i]);
551                 indexOfExponentialNumber++;
552                 numberOfExponentialNumber++;
553               }
554           }
555
556         } else {
557           try {
558             tokenSecondLine = tokenFirstSplit[0].split(" ");
559           } catch (NullPointerException e) {
560             return false;
561           }
562
563         }
564
565         indexOfExponentialNumber=0;
566
567         try {
568 //            System.out.format(result);
569
570         for (i=0; i<tokenSecondLine.length; i++) {
571 //            System.out.format("%d\n", i);
572
573           if (tokenSecondLine [i].equals("x")){
574             x[indexOfX] = Float.parseFloat(tokenSecondLine[i+2]);
575
576
577             if (i+3< tokenSecondLine.length
578                 && tokenSecondLine [i+3].equals("10")) {
579               x[indexOfX]*=Math.pow(10, exponentialNumber[
                       indexOfExponentialNumber]);
580               indexOfExponentialNumber++;
581             }
582             indexOfX++;
583             } else if (tokenSecondLine [i].equals("y")){
584             y[indexOfY] = Float.parseFloat(tokenSecondLine[i+2]);
585
586             if (i+3< tokenSecondLine.length
587                 && tokenSecondLine [i+3].equals("10")) {
588               y[indexOfY]*=Math.pow(10, exponentialNumber[
                       indexOfExponentialNumber]);
589               indexOfExponentialNumber++;
590             }
591             indexOfY++;
592
593             } else if (tokenSecondLine [i].equals("z")){
594             z[indexOfZ] = Float.parseFloat(tokenSecondLine[i+2]);
595
596             if (i+3 < tokenSecondLine.length
597                 && tokenSecondLine [i+3].equals("10")) {
598               z[indexOfZ]*=Math.pow(10, exponentialNumber[
                       indexOfExponentialNumber]);
599               indexOfExponentialNumber++;
600             }
601             indexOfZ++;
602             }
603
604         }
605         }catch (Exception e) {
606         e.printStackTrace();
607       }
```

```
608
609           v4[0].setXYZ((float) x[0],(float) y[0],(float) z[0]);
610       v4[1].setXYZ((float) x[1],(float) y[1],(float) z[1]);
611
612       return true;
613     }
614
615 }
```

```
 1 package com.drancom.programmableMatter.folding.controller;
 2
 3 public class Controller {
 4   void run() {
 5
 6   }
 7 }
```

```
 1 package com.drancom.programmableMatter.folding.dataFile;
 2
 3 import java.io.BufferedReader;
 4 import java.io.File;
 5 import java.io.FileInputStream;
 6 import java.io.FileNotFoundException;
 7 import java.io.FileOutputStream;
 8 import java.io.IOException;
 9
10 import com.drancom.programmableMatter.folding.controller.paper.Paper;
11
12 public class FileAngleData {
13   Paper[] papers;
14   String fileName;
15
16   public void build(String fileName, Paper[] papers){
17     int i;
18     int j;
19
20     int numberOfLines;
21
22     File file = new File(fileName);
23     try {
24       boolean success = file.createNewFile();
25       if (success) {
26         // File did not exist and was created
27       } else {
28
29       }
30
31     } catch (IOException e) {
32       // TODO Auto-generated catch block
33       e.printStackTrace();
34     }
35
36     String bufferLine;
37
38     this.fileName = fileName;
39     this.papers = papers;
40
41     FileOutputStream fileOutputStream = null;
42
43     try {
44       fileOutputStream = new FileOutputStream(file);
45     } catch (FileNotFoundException e) {
46       // TODO Auto-generated catch block
47       e.printStackTrace();
48     }
49
50     numberOfLines = papers[0].getNumberOfEdges();
51
```

```
52      for (i=0; i<numberOfLines; i++){
53        bufferLine = "";
54        bufferLine += papers[0].getLine(i).getAngle();
55        for (j=1; j<papers.length; j++) {
56          bufferLine += ", ";
57          bufferLine += papers[j].getLine(i).getAngle();
58        }
59
60        bufferLine += String.format("\n");
61
62        try {
63          fileOutputStream.write(bufferLine.getBytes());
64        } catch (IOException e) {
65          // TODO Auto-generated catch block
66          e.printStackTrace();
67        }
68      }
69
70      try {
71        fileOutputStream.close();
72      } catch (IOException e) {
73        // TODO Auto-generated catch block
74        e.printStackTrace();
75      }
76
77    }
78  }
```

```
1  package com.drancom.programmableMatter.folding.dataFile;
2
3  import java.io.BufferedReader;
4  import java.io.File;
5  import java.io.FileInputStream;
6  import java.io.IOException;
7  import java.io.InputStreamReader;
8
9
10 import com.drancom.programmableMatter.folding.controller.paper.Line;
11 import com.drancom.programmableMatter.folding.controller.paper.Paper;
12 import com.drancom.programmableMatter.folding.controller.paper.Point;
13
14 public class FileObj {
15
16   Paper paper;
17   String fileName;
18
19   public void load(String fileName, Paper paper){
20     File file = new File(fileName);
21
22     this.fileName = fileName;
23     this.paper = paper;
24
25     FileInputStream fileInputStream = null;
26         BufferedReader bufferedReader = null;
27
28         int index_v  = 0;
29         int index_vt = 0;
30         int index_e  = 0;
31
32         String [] token ;
33
34
35         String bufferLine;
36
37         if(file.exists()) {
38
39           try {
40             fileInputStream = new FileInputStream(file);
41             } catch (Exception e) {
```

```
42             e.printStackTrace();
43         }
44
45         bufferedReader = new BufferedReader(new InputStreamReader(
               fileInputStream));
46         while(true) {
47           // Load buffer
48           try {
49             bufferLine = bufferedReader.readLine();
50           } catch (IOException e) {
51             e.printStackTrace();
52         break;
53           }
54
55           //read command
56           try {
57             token = bufferLine.split(" ");
58           } catch (NullPointerException e) {
59
60             paper.build();
61
62             return;
63           }
64
65           if (token[0].equals("vt")){
66
67             Point temp_point;
68
69
70             temp_point = paper.getPoint(index_vt);
71
72             if (temp_point == null)
73             {
74               temp_point = new Point();
75             }
76
77             temp_point.setPointOnPaper( Float.parseFloat(token[1]),
78                         Float.parseFloat(token[2]),
79                         0.0f);
80
81             paper.setPoint(index_vt,temp_point);
82
83             index_vt++;
84
85           } else if (token[0].equals("#e")){
86
87           Line temp_line;
88           Point temp_startPoint;
89           Point temp_endPoint;
90           int index_startPoint;
91           int index_endPoint  ;
92           float temp_angle;
93           int temp_typeOfLine;
94
95           temp_line = paper.getLine(index_e);
96
97           if (temp_line == null)
98           {
99             temp_line = new Line();
100          }
101
102          index_startPoint = Integer.parseInt(token[1]) - 1;
103          index_endPoint   = Integer.parseInt(token[2]) - 1;
104
105          temp_startPoint = paper.getPoint(index_startPoint);
106
107          temp_endPoint = paper.getPoint(index_endPoint);
108
109          temp_typeOfLine = Integer.parseInt(token[3]);
```

```
110
111                 /**
112                  * TYPE_STATIC_LINE    = 0;
113                  * TYPE_EDGE_LINE      = 1;
114                  * TYPE_POSITIVE_LINE  = 2;
115                  * TYPE_NEGATIVE_LINE  = 3;
116                  */
117
118
119                 temp_angle      = Float.parseFloat(token[4]);
120
121                 if (temp_typeOfLine == Line.TYPE_POSITIVE_LINE) {
122                   temp_angle = (float) (  1 * (Math.PI / 180) * temp_angle);
123                 } else if (temp_typeOfLine == Line.TYPE_NEGATIVE_LINE)
124                 {
125                   temp_angle = (float) ( -1 * (Math.PI / 180) * temp_angle);
126                 } else {
127                   temp_angle = (float) (  1 * (Math.PI / 180) * temp_angle);
128                 }
129
130
131                 temp_line.setLine(paper,
132                         temp_startPoint,
133                         temp_endPoint,
134                         temp_angle,
135                         temp_typeOfLine);
136
137                 paper.setLine(index_e, temp_line);
138
139                 index_e++;
140
141                 }
142
143
144               }
145             }
146   }
147 }
```

```
 1 package com.drancom.programmableMatter.folding.dataFile;
 2
 3 import com.drancom.programmableMatter.folding.controller.paper.Paper;
 4 import com.drancom.programmableMatter.folding.origami.planner.Plan;
 5
 6 public interface FilePlan {
 7
 8    public void build(String fileName, Plan plan);
 9    public void build(String fileName, Paper[] papers);
10    public void build(String fileName, Plan plan, Paper[] papers);
11    public void read (String fileName, Plan plan);
12    public void read (String fileName, Paper[] papers);
13    public void read (String fileName, Plan plan, Paper[] papers);
14
15 }
```

```
 1 package com.drancom.programmableMatter.folding.dataFile;
 2
 3 import java.io.File;
 4 import java.io.FileNotFoundException;
 5 import java.io.FileOutputStream;
 6 import java.io.IOException;
 7
 8 import com.drancom.programmableMatter.folding.controller.paper.Paper;
 9 import com.drancom.programmableMatter.folding.origami.planner.Plan;
10
11 public class FilePlanForAngleActuration implements FilePlan{
12    Paper[] papers;
13    String fileName;
14
15    public void build(String fileName, Paper[] papers){
16      int i;
17      int time;
18      int j;
19      int k;
20      int l;
21
22      boolean ready;
23
24      int numberOfLines;
25
26      File file = new File(fileName);
27
28      try {
29        boolean success = file.createNewFile();
30        if (success) {
31          // File did not exist and was created
32        } else {
33        }
34
35      } catch (IOException e) {
36        // TODO Auto-generated catch block
37        e.printStackTrace();
38      }
39
40      String bufferLine = new String();
41      String bufferTime = new String();
42      String bufferLevel00 = new String();
43      String bufferLevel01 = new String();
44      String bufferLevel02 = new String();
45      String bufferLevel03 = new String();
46      String bufferLevel04 = new String();
47
48      String changeLine = String.format("\n");
49
50      this.fileName = fileName;
51      this.papers = papers;
52
53      FileOutputStream fileOutputStream = null;
54
55      try {
56        fileOutputStream = new FileOutputStream(file);
57      } catch (FileNotFoundException e) {
58        // TODO Auto-generated catch block
59        e.printStackTrace();
60      }
61
62      numberOfLines = papers[0].getNumberOfEdges();
63
64      // read paper
65      // set up the level.
66      // record the levels of edges when the level is change.
67
68      bufferLine = "time, level 1, level 2,  level -1, level -2, level  0" +
69           changeLine;
70
71      try {
72        fileOutputStream.write(bufferLine.getBytes());
73      } catch (IOException e) {
74        // TODO Auto-generated catch block
75        e.printStackTrace();
76      }
77
78      time = 0;
79      l = papers.length-1;
80      ready = false;
81      for (i = papers.length-1; i > 0; i-- ) {
82        l = i;
```

```
82          for (j = 0; j < numberOfLines ; j++ ) {
83            if (papers[i     ].getLine(j).getLevelOfActuratingPower()!= 0.0f) {
84              ready = true;
85              break;
86            }
87          }
88
89          if (ready){
90            break;
91
92          }
93        }
94
95        bufferTime = Integer.toString(i);
96        bufferLevel00 = "";
97        bufferLevel01 = "";
98        bufferLevel02 = "";
99        bufferLevel03 = "";
100       bufferLevel04 = "";
101
102       for (k = 0; k < numberOfLines ; k++) {
103
104         if (papers[i].getLine(k).getLevelOfActuratingPower() == 2f) {
105           bufferLevel02 += "f" + k + " ";
106         } else if (papers[i].getLine(k).getLevelOfActuratingPower() == -2f) {
107           bufferLevel04 += "f" + k + " ";
108         } else if (papers[i].getLine(k).getLevelOfActuratingPower() > 0.0f) {
109           bufferLevel01 += "f" + k + " ";
110         } else if (papers[i].getLine(k).getLevelOfActuratingPower() < 0.0f) {
111           bufferLevel03 += "f" + k + " ";
112         } else {
113           bufferLevel00 += "f" + k + " ";
114         }
115       }
116
117       bufferTime = Integer.toString(time);
118
119       bufferLine = bufferTime + ", ";
120       bufferLine += bufferLevel01 + ", ";
121       bufferLine += bufferLevel02 + ", ";
122       bufferLine += bufferLevel03 + ", ";
123       bufferLine += bufferLevel04 + ", ";
124 //      bufferLine += bufferLevel00;
125       bufferLine += changeLine;
126
127       try {
128         fileOutputStream.write(bufferLine.getBytes());
129       } catch (IOException e) {
130         // TODO Auto-generated catch block
131         e.printStackTrace();
132       }
133
134
135       time++;
136
137       for (i = l-1; i > 1; i-- ) {
138
139         for (j = 0; j < numberOfLines ; j++ ) {
140           if (Math.abs(papers[i     ].getLine(j).getLevelOfActuratingPower())
                  !=
141             Math.abs(papers[i + 1].getLine(j).getLevelOfActuratingPower()) )
                  {
142             // lavel  1, lavel  2, lavel -1, lavel -2
143
144             bufferTime = Integer.toString(i);
145             bufferLevel00 = "";
146             bufferLevel01 = "";
147             bufferLevel02 = "";
148             bufferLevel03 = "";
149             bufferLevel04 = "";
150
151             for (k = 0; k < numberOfLines ; k++) {
152
153
154               if (papers[i].getLine(k).getLevelOfActuratingPower() == 2f) {
155                 bufferLevel02 += "f" + k + " ";
156               } else if (papers[i].getLine(k).getLevelOfActuratingPower() ==
                      -2f) {
157                 bufferLevel04 += "f" + k + " ";
158               } else if (papers[i].getLine(k).getLevelOfActuratingPower() >
                      0.0f) {
159                 bufferLevel01 += "f" + k + " ";
160               } else if (papers[i].getLine(k).getLevelOfActuratingPower() <
                      0.0f) {
161                 bufferLevel03 += "f" + k + " ";
162               } else {
163                 bufferLevel00 += "f" + k + " ";
164               }
165             }
166
167
168             bufferTime = Integer.toString(time);
169
170             bufferLine = bufferTime + ", ";
171             bufferLine += bufferLevel01 + ", ";
172             bufferLine += bufferLevel02 + ", ";
173             bufferLine += bufferLevel03 + ", ";
174             bufferLine += bufferLevel04 + ", ";
175 //            bufferLine += bufferLevel00;
176             bufferLine += changeLine;
177
178             try {
179               fileOutputStream.write(bufferLine.getBytes());
180             } catch (IOException e) {
181               // TODO Auto-generated catch block
182               e.printStackTrace();
183             }
184             break;
185           }
186         }
187         time++;
188       }
189
190       try {
191         fileOutputStream.close();
192       } catch (IOException e) {
193         // TODO Auto-generated catch block
194         e.printStackTrace();
195       }
196     }
197
198     @Override
199     public void build(String fileName, Plan plan) {
200       // TODO Auto-generated method stub
201
202     }
203
204     @Override
205     public void read(String fileName, Plan plan) {
206       // TODO Auto-generated method stub
207
208     }
209
210     @Override
211     public void read(String fileName, Paper[] papers) {
212       // TODO Auto-generated method stub
213
214     }
```

```
215
216   @Override
217   public void build(String fileName, Plan plan, Paper[] papers) {
218     // TODO Auto-generated method stub
219
220   }
221
222   @Override
223   public void read(String fileName, Plan plan, Paper[] papers) {
224     // TODO Auto-generated method stub
225
226   }
227 }
```

```
 1 package com.drancom.programmableMatter.folding.dataFile;
 2
 3 import java.io.File;
 4 import java.io.FileNotFoundException;
 5 import java.io.FileOutputStream;
 6 import java.io.IOException;
 7
 8 import com.drancom.programmableMatter.folding.controller.paper.Paper;
 9 import com.drancom.programmableMatter.folding.origami.planner.Plan;
10 import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
       ;
11
12 public class FilePlanForWiring implements FilePlan {
13
14
15   public void build(String fileName, Plan plan){
16     int i;
17     int j;
18     int k;
19     int l;
20
21     int time;
22
23     boolean ready;
24
25     int numberOfLines;
26
27     File file = new File(fileName);
28     PlanForWiring planForWiring = (PlanForWiring) plan;
29     boolean [][][][][] planTable = planForWiring.getPlanTable();
30
31     try {
32       boolean success = file.createNewFile();
33       if (success) {
34         // File did not exist and was created
35       } else {
36       }
37
38     } catch (IOException e) {
39       // TODO Auto-generated catch block
40       e.printStackTrace();
41     }
42
43     String bufferTitle = new String();
44     String bufferLine  = new String();
45     String changeLine = String.format("\n");
46
47     FileOutputStream fileOutputStream = null;
48
49     try {
50       fileOutputStream = new FileOutputStream(file);
51     } catch (FileNotFoundException e) {
52       // TODO Auto-generated catch block
53       e.printStackTrace();
54     }
55
56     // read paper
57     // set up the level.
58     // record the levels of edges when the level is change.
59
60     bufferTitle = "[folding][phases][inside][edgeNumber][activation]";
61     bufferLine = bufferTitle;
62     bufferLine += changeLine;
63
64     try {
65       fileOutputStream.write(bufferLine.getBytes());
66     } catch (IOException e) {
67       // TODO Auto-generated catch block
68       e.printStackTrace();
69     }
70
71     // planTable [folding][phases][inside][edgeNumber][activation]
72     for (i = 0; i < 2 ; i++) {
73       for (j = 0; j < planForWiring.getNumberOfPhases(); j++) {
74         for (k = 0; k < 2 ; k++ ) {
75           for (l = 0 ; l < planForWiring.getNumberOfEdges(); l++ ) {
76
77             bufferLine = String.format("%d, %d, %d, %d, %s", i, j, k, l,
                   Boolean.toString(planTable[i][j][k][l]));
78             bufferLine += changeLine;
79
80             // print to file
81             try {
82               fileOutputStream.write(bufferLine.getBytes());
83             } catch (IOException e) {
84               // TODO Auto-generated catch block
85               e.printStackTrace();
86             }
87           }
88         }
89       }
90     }
91
92     try {
93       fileOutputStream.close();
94     } catch (IOException e) {
95       // TODO Auto-generated catch block
96       e.printStackTrace();
97     }
98   }
99
100  @Override
101  public void build(String fileName, Paper[] papers) {
102    // TODO Auto-generated method stub
103
104  }
105
106  @Override
107  public void read(String fileName, Plan plan) {
108    // TODO Auto-generated method stub
109
110  }
111
112  @Override
113  public void read(String fileName, Paper[] papers) {
114    // TODO Auto-generated method stub
115
116  }
117
118  @Override
119  public void build(String fileName, Plan plan, Paper[] papers) {
120    // TODO Auto-generated method stub
121
122  }
```

```
123
124     @Override
125     public void read(String fileName, Plan plan, Paper[] papers) {
126       // TODO Auto-generated method stub
127
128     }
129 }
```

```
1  package com.drancom.programmableMatter.folding.dataFile;
2
3  import java.io.File;
4  import java.io.FileInputStream;
5
6  public class FilePmf {
7    void load(String fileName){
8      File file = new File(fileName);
9
10         if(file.exists()) {
11            FileInputStream fileInputStream = null;
12            try {
13              fileInputStream = new FileInputStream(file);
14            } catch (Exception e) {
15
16            }
17         }
18   }
19 }
```

```
1  package com.drancom.programmableMatter.folding.dataFile;
2
3  public class Transform {
4
5  }
```

```
1  package com.drancom.programmableMatter.folding.monitor;
2
3
4  import java.awt.Dimension;
5  import java.awt.Frame;
6  import java.awt.event.*;
7
8  import javax.media.opengl.*;
9
10 import sun.text.normalizer.UProperty;
11
12 import com.drancom.programmableMatter.folding.controller.paper.Paper;
13 import com.drancom.programmableMatter.folding.controller.paper.Point;
14 import com.drancom.programmableMatter.folding.controller.paper.Polygon;
15 import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
16 import com.sun.opengl.util.*;
17
18 public class MainWindow implements GLEventListener, MouseListener,
       MouseMotionListener {
19   Paper[] papers;
20
21   // light
22   // float pos0[] = { -100.0f, 130.0f, 150.0f, 1.0f };
23   public final static float pos0[] = { -100.0f, 100.0f, 100.0f, 1.0f };
24   public final static float ambientLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
25   public final static float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
26   public final static float specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};
27
28   public final static float WRITE[]  = { 1.0f, 1.0f, 1.0f, 0.2f };
29   public final static float RED[]    = { 1.0f, 0.0f, 0.0f, 1.0f };
30   public final static float GREEN[]  = { 0.0f, 1.0f, 0.0f, 1.0f };
31   public final static float YELLOW[] = { 1.0f, 1.0f, 0.0f, 1.0f };
32   public final static float BLUE[]   = { 0.0f, 0.0f, 1.0f, 0.0f };
```

```
33   public final static float BLACK[]  = { 0.0f, 0.0f, 0.0f, 1.0f };
34
35   final static int SPEED_OF_ANIMATION = 15; //10 is default
36   int counterForSpeedOfAnimation = 0;
37
38   boolean isAnimating=false;
39
40   public void run(Paper[] papers) {
41     this.papers = papers;
42     paperGlId = new int[papers.length];
43
44     Frame frame = new Frame("Play Window - Programmable Matter by Folding")
           ;
45       GLCanvas canvas = new GLCanvas();
46
47     canvas.addGLEventListener(this);
48     frame.add(canvas);
49     frame.setSize(800, 800);
50     final Animator animator = new Animator(canvas);
51     frame.addWindowListener(new WindowAdapter() {
52       public void windowClosing(WindowEvent e) {
53             // Run this on another thread than the AWT event queue to
54         // make sure the call to Animator.stop() completes before
55         // exiting
56         new Thread(new Runnable() {
57           public void run() {
58             animator.stop();
59             System.exit(0);
60           }
61         }).start();
62       }
63     });
64       frame.show();
65       animator.start();
66   }
67
68   private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;
69
70   private int paperGlId[];
71   private int currentIndex_PaperGlId;
72
73   private boolean isFolding = true;
74
75   public boolean isFolding() {
76     return isFolding;
77   }
78
79   private int prevMouseX, prevMouseY;
80   private boolean mouseRButtonDown = false;
81
82   public void init(GLAutoDrawable drawable) {
83     int i;
84
85     // Use debug pipeline
86     // drawable.setGL(new DebugGL(drawable.getGL()));
87
88     GL gl = drawable.getGL();
89
90     System.err.println("INIT GL IS: " + gl.getClass().getName());
91     System.err.println("Chosen GLCapabilities: " + drawable.
           getChosenGLCapabilities());
92
93     gl.setSwapInterval(1);
94
95     // Blend
96     gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE);
97     gl.glEnable(GL.GL_BLEND);
98
99
```

```
100 //     gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
101 //       gl.glClearDepth(1.0f);
102
103       gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos0, 0);
104       gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambientLight, 0);
105       gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuseLight, 0);
106       gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular, 0);
107
108       gl.glEnable(GL.GL_CULL_FACE);
109
110       gl.glEnable(GL.GL_LIGHTING);
111         gl.glEnable(GL.GL_LIGHT0);
112
113 //      gl.glEnable(GL.GL_DEPTH_TEST);
114
115       /* make the papers */
116       for (i=0; i<papers.length; i++){
117         paperGlId[i] = gl.glGenLists(1);
118         gl.glNewList(paperGlId[i], GL.GL_COMPILE);
119         buildGlPaper(gl, papers[i]);
120         gl.glEndList();
121       }
122
123     // currentIndex_PaperGlId = 0;
124       currentIndex_PaperGlId = papers.length -1;
125
126   //  gl.glEnable(GL.GL_NORMALIZE);
127
128       drawable.addMouseListener(this);
129       drawable.addMouseMotionListener(this);
130   }
131
132   public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
          height) {
133     GL gl = drawable.getGL();
134
135     float h = (float)height / (float)width;
136
137     gl.glMatrixMode(GL.GL_PROJECTION);
138
139     System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
140     System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
141     System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
142     gl.glLoadIdentity();
143     gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
144     gl.glMatrixMode(GL.GL_MODELVIEW);
145     gl.glLoadIdentity();
146     gl.glTranslatef(0.0f, 0.0f, -40.0f);
147   }
148
149   public void display(GLAutoDrawable drawable) {
150     // Turn the gears' teeth
151
152     // Get the GL corresponding to the drawable we are animating
153     GL gl = drawable.getGL();
154
155     // Special handling for the case where the GLJPanel is translucent
156     // and wants to be composited with other Java 2D content
157     if ((drawable instanceof GLJPanel) &&
158         !((GLJPanel) drawable).isOpaque() &&
159         ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
160       gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
161     } else {
162       gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
163     }
164
165
166     // Rotate the entire assembly of gears based on how the user
167     // dragged the mouse around
168     gl.glPushMatrix();
169     gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
170     gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
171     gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);
172
173     // Place the first gear and call its display list
174     gl.glPushMatrix();
175       gl.glTranslatef(-5.0f, -5.0f, 0.0f);
176       gl.glCallList(paperGlId[((int) currentIndex_PaperGlId)]);
177     gl.glPopMatrix();
178
179
180     // Remember that every push needs a pop; this one is paired with
181     // rotating the entire gear assembly
182     gl.glPopMatrix();
183     if (isAnimating == true){
184       if (isFolding() == true ){
185         counterForSpeedOfAnimation--;
186
187         if (counterForSpeedOfAnimation <= 0) {
188           counterForSpeedOfAnimation = SPEED_OF_ANIMATION;
189
190           currentIndex_PaperGlId--;
191
192           if (currentIndex_PaperGlId < 0) {
193             currentIndex_PaperGlId = 0;
194             isFolding = false;
195             isAnimating = false;
196           }
197         }
198       } else {
199         counterForSpeedOfAnimation++;
200         if (counterForSpeedOfAnimation >= SPEED_OF_ANIMATION) {
201           counterForSpeedOfAnimation = 0;
202
203           currentIndex_PaperGlId++;
204
205           if (currentIndex_PaperGlId >= papers.length - 1 ) {
206             currentIndex_PaperGlId = papers.length- 1;
207             isFolding = true;
208             isAnimating = false;
209           }
210         }
211       }
212     }
213   }
214
215   public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
          boolean deviceChanged) {}
216
217   public void buildGlPaper(GL gl, Paper paper){
218     int i;
219     int j;
220     int numberOfLine = paper.getNumberOfEdges();
221     int numberOfPolygon = paper.getNumberOfPolygons();
222
223     Polygon polygon;
224     Point[] polygonPoints;
225
226     Vector startPointVector;
227     Vector endPointVector ;
228
229     gl.glShadeModel(GL.GL_FLAT);
230
231     gl.glNormal3f(0.0f, 0.0f, 1.0f);
232
233     /* draw polygon */
234     numberOfPolygon = paper.getNumberOfPolygons();
235
```

```
236     for (i=0; i<numberOfPolygon; i++) {
237       polygon = paper.getPolygon(i);
238
239       polygonPoints = polygon.getPoints();
240
241       gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WRITE, 0);
242
243       gl.glBegin(GL.GL_TRIANGLES);
244         for (j=0; j<3; j++){
245           gl.glVertex3d(polygonPoints[j].getXInReal() * 8
246                , polygonPoints[j].getYInReal() * 8
247                , polygonPoints[j].getZInReal() * 8 );
248
249         }
250         for (j=2; j>=0; j--){
251         gl.glVertex3d(polygonPoints[j].getXInReal() * 8
252                , polygonPoints[j].getYInReal() * 8
253                , polygonPoints[j].getZInReal() * 8 );
254
255         }
256
257       gl.glEnd();
258
259
260     }
261
262     /* draw lines */
263     numberOfLine = paper.getNumberOfEdges();
264
265     for (i=0; i<numberOfLine; i++) {
266
267         gl.glLineWidth(1.0f);
268
269       startPointVector = paper.getLine(i).getStartPoint().getVectorInReal()
              ;
270       endPointVector = paper.getLine(i).getEndPoint().getVectorInReal();
271
272
273       if(paper.getLine(i).getLevelOfActuratingPower() > 0.01) {
274
275         // actuating
276         gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, RED, 0);
277
278       }else if (paper.getLine(i).getLevelOfActuratingPower() < -0.01){
279
280         // passive moving
281         gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, BLUE, 0);
282
283       }else {
284
285         // stop
286         gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WRITE, 0);
287       }
288
289
290       gl.glBegin(GL.GL_LINES);
291
292       gl.glVertex3f((float)startPointVector.getX()*8,
293           (float)startPointVector.getY()*8,
294           (float)startPointVector.getZ()*8);
295
296       gl.glVertex3f((float)endPointVector.getX()*8,
297           (float)endPointVector.getY()*8,
298           (float)endPointVector.getZ()*8);
299       gl.glEnd();
300
301
302     }
303   }
```

```
304
305     // Methods required for the implementation of MouseListener
306   public void mouseEntered(MouseEvent e) {}
307   public void mouseExited(MouseEvent e) {}
308   public void mousePressed(MouseEvent e) {
309       prevMouseX = e.getX();
310       prevMouseY = e.getY();
311       if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
312         mouseRButtonDown = true;
313       }
314
315       if (mouseRButtonDown == true) {
316         isAnimating = true;
317       }
318
319
320   }
321
322   public void mouseReleased(MouseEvent e) {
323     if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
324       mouseRButtonDown = false;
325       }
326
327   }
328
329   public void mouseClicked(MouseEvent e) {}
330
331   // Methods required for the implementation of MouseMotionListener
332   public void mouseDragged(MouseEvent e) {
333     if (mouseRButtonDown == false) {
334       int x = e.getX();
335       int y = e.getY();
336       Dimension size = e.getComponent().getSize();
337
338       float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
339       float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);
340
341       prevMouseX = x;
342       prevMouseY = y;
343
344       view_rotx += thetaX;
345       view_roty += thetaY;
346     }else {
347
348     }
349   }
350
351   public void mouseMoved(MouseEvent e) {
352
353   }
354 }
```

```
 1 package com.drancom.programmableMatter.folding.monitor;
 2
 3
 4
 5 import java.awt.Dimension;
 6 import java.awt.Frame;
 7 import java.awt.event.MouseEvent;
 8 import java.awt.event.MouseListener;
 9 import java.awt.event.MouseMotionListener;
10 import java.awt.event.WindowAdapter;
11 import java.awt.event.WindowEvent;
12
13 import javax.media.opengl.*;
14
15 import com.drancom.programmableMatter.folding.controller.paper.Paper;
16 import com.drancom.programmableMatter.folding.controller.paper.Point;
17 import com.drancom.programmableMatter.folding.controller.paper.Polygon;
```

```
18 import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
19 import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
       ;
20 import com.drancom.programmableMatter.folding.origami.planner.
       PlannerForWiring;
21 import com.sun.opengl.util.Animator;
22
23
24 public class MainWindowForFoldingRobotWiring  implements GLEventListener,
       MouseListener, MouseMotionListener {
25   PlanForWiring planForWiring;
26   boolean [][][][] planOfTable;
27   Paper[] papers;
28
29   // light
30   // float pos0[] =  {  -100.0f,  180.0f, 150.0f,  1.0f };
31   float pos0[] =  {  -100.0f,  100.0f, 100.0f,  1.0f };
32   float ambientLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
33   float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
34   float specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};
35
36
37
38   float white[] = {  1.0f,  1.0f,  1.0f,  0.2f };
39   float red[] =   {  1.0f,  0.0f,  0.0f,  1.0f };
40   float green[] = {  0.0f,  1.0f,  0.0f,  1.0f };
41   float yellow[] = {  1.0f,  1.0f,  0.0f,  1.0f };
42   float blue[] = { 0.0f, 0.0f, 1.0f, 0.0f };
43   float black[] = { 0.0f, 0.0f, 0.0f, 1.0f };
44
45   final static int SPEED_OF_ANIMATION = 15; //10 is default
46   int counterForSpeedOfAnimation = 0;
47
48   boolean isAnimating=false;
49
50   public void run(Paper[] papers, PlanForWiring planForWiring) {
51     this.papers = papers;
52     paperGlId = new int[papers.length];
53
54     this.planForWiring = planForWiring;
55     planOfTable = planForWiring.getPlanTable();
56
57     Frame frame = new Frame("Play Window - Programmable Matter by Folding")
         ;
58       GLCanvas canvas = new GLCanvas();
59
60     canvas.addGLEventListener(this);
61     frame.add(canvas);
62     frame.setSize(800, 800);
63     final Animator animator = new Animator(canvas);
64     frame.addWindowListener(new WindowAdapter() {
65       public void windowClosing(WindowEvent e) {
66             // Run this on another thread than the AWT event queue to
67         // make sure the call to Animator.stop() completes before
68         // exiting
69         new Thread(new Runnable() {
70           public void run() {
71             animator.stop();
72             System.exit(0);
73           }
74         }).start();
75           }
76     });
77       frame.show();
78       animator.start();
79   }
80
81   private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;
82
83   private int paperGlId[];
84   private int currentIndex_PaperGlId;
85
86   private boolean isFolding = true;
87
88   public boolean isFolding() {
89     return isFolding;
90   }
91
92   private int prevMouseX, prevMouseY;
93   private boolean mouseRButtonDown = false;
94
95   public void init(GLAutoDrawable drawable) {
96     int i;
97
98     // Use debug pipeline
99     // drawable.setGL(new DebugGL(drawable.getGL()));
100
101    GL gl = drawable.getGL();
102
103    System.err.println("INIT GL IS: " + gl.getClass().getName());
104    System.err.println("Chosen GLCapabilities: " + drawable.
         getChosenGLCapabilities());
105
106    gl.setSwapInterval(1);
107
108    // Blend
109    gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE);
110    gl.glEnable(GL.GL_BLEND);
111
112
113 //     gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
114 //       gl.glClearDepth(1.0f);
115
116    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos0, 0);
117    gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambientLight, 0);
118    gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuseLight, 0);
119    gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular, 0);
120
121    // gl.glEnable(GL.GL_CULL_FACE);
122
123    gl.glEnable(GL.GL_LIGHTING);
124      gl.glEnable(GL.GL_LIGHT0);
125
126 //    gl.glEnable(GL.GL_DEPTH_TEST);
127
128      /* make the papers */
129      for (i=0; i<papers.length; i++){
130        paperGlId[i] = gl.glGenLists(1);
131        gl.glNewList(paperGlId[i], GL.GL_COMPILE);
132        buildGlPaper(gl, papers[i]);
133        gl.glEndList();
134      }
135
136    // currentIndex_PaperGlId = 0;
137      currentIndex_PaperGlId = papers.length -1;
138
139  //  gl.glEnable(GL.GL_NORMALIZE);
140
141      drawable.addMouseListener(this);
142      drawable.addMouseMotionListener(this);
143  }
144
145  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
       height) {
146    GL gl = drawable.getGL();
147
148    float h = (float)height / (float)width;
149
```

```
150       gl.glMatrixMode(GL.GL_PROJECTION);
151
152       System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
153       System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
154       System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
155       gl.glLoadIdentity();
156       gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
157       gl.glMatrixMode(GL.GL_MODELVIEW);
158       gl.glLoadIdentity();
159       gl.glTranslatef(0.0f, 0.0f, -40.0f);
160     }
161
162     public void display(GLAutoDrawable drawable) {
163       // Turn the gears' teeth
164
165       // Get the GL corresponding to the drawable we are animating
166       GL gl = drawable.getGL();
167
168       // Special handling for the case where the GLJPanel is translucent
169       // and wants to be composited with other Java 2D content
170       if ((drawable instanceof GLJPanel) &&
171           !((GLJPanel) drawable).isOpaque() &&
172           ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
173         gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
174       } else {
175         gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
176       }
177
178
179       // Rotate the entire assembly of gears based on how the user
180       // dragged the mouse around
181       gl.glPushMatrix();
182       gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
183       gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
184       gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);
185
186       // Place the first gear and call its display list
187       gl.glPushMatrix();
188         gl.glTranslatef(-5.0f, -5.0f, 0.0f);
189         gl.glCallList(paperGlId[((int) currentIndex_PaperGlId)]);
190       gl.glPopMatrix();
191
192
193       // Remember that every push needs a pop; this one is paired with
194       // rotating the entire gear assembly
195       gl.glPopMatrix();
196       if (isAnimating == true){
197         if (isFolding() == true ){
198           counterForSpeedOfAnimation--;
199
200           if (counterForSpeedOfAnimation <= 0) {
201             counterForSpeedOfAnimation = SPEED_OF_ANIMATION;
202
203             currentIndex_PaperGlId--;
204
205             if (currentIndex_PaperGlId < 0) {
206               currentIndex_PaperGlId = 0;
207               isFolding = false;
208               isAnimating = false;
209             }
210           }
211         } else {
212           counterForSpeedOfAnimation++;
213           if (counterForSpeedOfAnimation >= SPEED_OF_ANIMATION) {
214             counterForSpeedOfAnimation = 0;
215
216             currentIndex_PaperGlId++;
217
218             if (currentIndex_PaperGlId >= papers.length - 1 ) {
219               currentIndex_PaperGlId = papers.length- 1;
220               isFolding = true;
221               isAnimating = false;
222             }
223           }
224         }
225       }
226     }
227
228     public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
          boolean deviceChanged) {}
229
230     public void buildGlPaper(GL gl, Paper paper){
231
232       // for phase 0 and 1
233       int i;
234       int j;
235       int numberOfLine = paper.getNumberOfEdges();
236       int numberOfPolygon = paper.getNumberOfPolygons();
237
238       Polygon polygon;
239       Point[] polygonPoints;
240
241       Vector startPointVector;
242       Vector endPointVector ;
243
244       gl.glShadeModel(GL.GL_FLAT);
245
246       gl.glNormal3f(0.0f, 0.0f, 1.0f);
247
248       /* draw polygon */
249       numberOfPolygon = paper.getNumberOfPolygons();
250
251       for (i=0; i<numberOfPolygon; i++) {
252         polygon = paper.getPolygon(i);
253
254         polygonPoints = polygon.getPoints();
255
256         gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);
257
258         gl.glBegin(GL.GL_TRIANGLES);
259           for (j=0; j<3; j++){
260             gl.glVertex3d(polygonPoints[j].getXInReal() * 8
261                 , polygonPoints[j].getYInReal() * 8
262                 , polygonPoints[j].getZInReal() * 8 );
263
264           }
265 //          for (j=2; j>=0; j--){
266 //            gl.glVertex3f(polygonPoints[j].getXInReal() * 8
267 //                , polygonPoints[j].getYInReal() * 8
268 //                , polygonPoints[j].getZInReal() * 8 );
269 //
270 //          }
271
272         gl.glEnd();
273
274
275       }
276
277       /* draw lines */
278       numberOfLine = paper.getNumberOfEdges();
279
280       for (i=0; i<numberOfLine; i++) {
281
282           gl.glLineWidth(1.0f);
283
284         startPointVector = paper.getLine(i).getStartPoint().getVectorInReal()
              ;
285         endPointVector = paper.getLine(i).getEndPoint().getVectorInReal();
```

```
286
287       /**/
288
289         if(planOfTable[0][0][0][i] == true) {
290
291           // actuating
292           gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, red, 0);
293
294         }else if (planOfTable[0][0][1][i] == true){
295
296           // passive moving
297           gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, blue, 0);
298
299         }else {
300
301           // stop
302           gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);
303         }
304 /** /
305         if(paper.getLine(i).getLevelOfActuratingPower() > 0.01) {
306
307           // actuating
308           gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, red, 0);
309
310         }else if (paper.getLine(i).getLevelOfActuratingPower() < -0.01){
311
312           // passive moving
313           gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, blue, 0);
314
315         }else {
316
317           // stop
318           gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);
319         }
320 /**/
321         gl.glBegin(GL.GL_LINES);
322
323         gl.glVertex3f((float)startPointVector.getX()*8,
324             (float)startPointVector.getY()*8,
325             (float)startPointVector.getZ()*8);
326
327         gl.glVertex3f((float)endPointVector.getX()*8,
328             (float)endPointVector.getY()*8,
329             (float)endPointVector.getZ()*8);
330         gl.glEnd();
331
332
333       }
334     }
335
336       // Methods required for the implementation of MouseListener
337     public void mouseEntered(MouseEvent e) {}
338     public void mouseExited(MouseEvent e) {}
339     public void mousePressed(MouseEvent e) {
340         prevMouseX = e.getX();
341         prevMouseY = e.getY();
342         if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
343           mouseRButtonDown = true;
344         }
345
346         if (mouseRButtonDown == true) {
347           isAnimating = true;
348         }
349
350
351     }
352
353     public void mouseReleased(MouseEvent e) {
354
355       if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
356         mouseRButtonDown = false;
357         }
358
359     }
360
361     public void mouseClicked(MouseEvent e) {}
362
363     // Methods required for the implementation of MouseMotionListener
364     public void mouseDragged(MouseEvent e) {
365       if (mouseRButtonDown == false) {
366         int x = e.getX();
367         int y = e.getY();
368         Dimension size = e.getComponent().getSize();
369
370         float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
371         float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);
372
373         prevMouseX = x;
374         prevMouseY = y;
375
376         view_rotx += thetaX;
377         view_roty += thetaY;
378       }else {
379
380       }
381     }
382
383     public void mouseMoved(MouseEvent e) {
384
385     }
386
387 }
```

```
1  package com.drancom.programmableMatter.folding.monitor;
2
3
4
5  import java.awt.Dimension;
6  import java.awt.Frame;
7  import java.awt.event.MouseEvent;
8  import java.awt.event.MouseListener;
9  import java.awt.event.MouseMotionListener;
10 import java.awt.event.WindowAdapter;
11 import java.awt.event.WindowEvent;
12 import java.util.ArrayList;
13
14 import javax.media.opengl.*;
15
16 import com.drancom.programmableMatter.folding.controller.paper.Line;
17 import com.drancom.programmableMatter.folding.controller.paper.Paper;
18 import com.drancom.programmableMatter.folding.controller.paper.Point;
19 import com.drancom.programmableMatter.folding.controller.paper.Polygon;
20
21 import com.sun.opengl.util.Animator;
22
23
24 public class MonitorOfPaperArray  implements GLEventListener, MouseListener
       , MouseMotionListener {
25   public final static float ZOOM_MAGNIFICATION = 8.0f;
26   public final static float LINEWIDTH = 2.0f;
27
28   ArrayList<Paper> paperArray;
29
30   // light
31   // float pos0[] =  {  -100.0f,  130.0f, 150.0f,  1.0f };
32   float pos0[] =  {  -100.0f,  100.0f, 100.0f,  1.0f };
33   float ambientLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
34   float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
```

```
35    float specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};
36
37    float white[] = {  1.0f,  1.0f,  1.0f,  0.2f };
38    float red[] =   {  1.0f,  0.0f,  0.0f,  1.0f };
39    float green[] = {  0.0f,  1.0f,  0.0f,  1.0f };
40    float yellow[] = {  1.0f,  1.0f,  0.0f,  1.0f };
41    float blue[] = { 0.0f, 0.0f, 1.0f, 0.0f };
42    float black[] = { 0.0f, 0.0f, 0.0f, 1.0f };
43
44    final static int SPEED_OF_ANIMATION = 15; //10 is default
45    int counterForSpeedOfAnimation = 0;
46
47    boolean isAnimating = false;
48
49
50    int numberOfPapers;
51
52    Frame frame;
53
54
55
56    public void run(ArrayList<Paper> paperArray) {
57      numberOfPapers = paperArray.size();
58
59      this.paperArray = paperArray;
60
61      paperGlId = new int[numberOfPapers];
62
63      frame = new Frame("Monitor Of Unfolding Algorithm - Programmable Matter
            by Folding");
64        GLCanvas canvas = new GLCanvas();
65
66      canvas.addGLEventListener(this);
67      frame.add(canvas);
68      frame.setSize(800, 800);
69      final Animator animator = new Animator(canvas);
70      frame.addWindowListener(new WindowAdapter() {
71        public void windowClosing(WindowEvent e) {
72              // Run this on another thread than the AWT event queue to
73          // make sure the call to Animator.stop() completes before
74          // exiting
75          new Thread(new Runnable() {
76            public void run() {
77              animator.stop();
78              System.exit(0);
79            }
80          }).start();
81            }
82      });
83        frame.show();
84        animator.start();
85    }
86
87
88    private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;
89
90    private int paperGlId[];
91    private int currentIndex_PaperGlId;
92
93    private boolean isFolding = true;
94
95    public boolean isFolding() {
96      return isFolding;
97    }
98
99    private int prevMouseX, prevMouseY;
100   private boolean mouseRButtonDown = false;
101
102   public void init(GLAutoDrawable drawable) {
103     int i;
104
105     // Use debug pipeline
106     // drawable.setGL(new DebugGL(drawable.getGL()));
107
108     GL gl = drawable.getGL();
109
110     System.err.println("INIT GL IS: " + gl.getClass().getName());
111     System.err.println("Chosen GLCapabilities: " + drawable.
            getChosenGLCapabilities());
112
113     gl.setSwapInterval(1);
114
115     // Blend
116     gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE);
117     gl.glEnable(GL.GL_BLEND);
118
119
120 //     gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
121 //       gl.glClearDepth(1.0f);
122
123     gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos0, 0);
124     gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambientLight, 0);
125     gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuseLight, 0);
126     gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular, 0);
127
128     //gl.glEnable(GL.GL_CULL_FACE);
129
130     gl.glEnable(GL.GL_LIGHTING);
131       gl.glEnable(GL.GL_LIGHT0);
132
133 //     gl.glEnable(GL.GL_DEPTH_TEST);
134
135       /* make the papers */
136       for (i=0; i < numberOfPapers; i++){
137         paperGlId[i] = gl.glGenLists(1);
138         gl.glNewList(paperGlId[i], GL.GL_COMPILE);
139           buildGlPaper(gl, i, paperArray.get(i));
140         gl.glEndList();
141       }
142
143     currentIndex_PaperGlId = 0;
144
145
146 //     gl.glEnable(GL.GL_NORMALIZE);
147
148       drawable.addMouseListener(this);
149       drawable.addMouseMotionListener(this);
150   }
151
152   public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
          height) {
153     GL gl = drawable.getGL();
154
155     float h = (float)height / (float)width;
156
157     gl.glMatrixMode(GL.GL_PROJECTION);
158
159     System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
160     System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
161     System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
162     gl.glLoadIdentity();
163     gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
164     gl.glMatrixMode(GL.GL_MODELVIEW);
165     gl.glLoadIdentity();
166     gl.glTranslatef(0.0f, 0.0f, -40.0f);
167   }
168
169   public void display(GLAutoDrawable drawable) {
```

```
170     // Turn the gears' teeth
171
172     // Get the GL corresponding to the drawable we are animating
173     GL gl = drawable.getGL();
174
175     // Special handling for the case where the GLJPanel is translucent
176     // and wants to be composited with other Java 2D content
177     if ((drawable instanceof GLJPanel) &&
178         !((GLJPanel) drawable).isOpaque() &&
179         ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
180       gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
181     } else {
182       gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
183     }
184
185
186     // Rotate the entire assembly of gears based on how the user
187     // dragged the mouse around
188     gl.glPushMatrix();
189     gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
190     gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
191     gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);
192
193     // Place the first gear and call its display list
194     gl.glPushMatrix();
195       gl.glTranslatef(-0.0f, -0.0f, 0.0f);
196       gl.glCallList(paperGlId[((int) currentIndex_PaperGlId)]);
197     gl.glPopMatrix();
198
199
200     // Remember that every push needs a pop; this one is paired with
201     // rotating the entire gear assembly
202     gl.glPopMatrix();
203
204   }
205
206   public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
          boolean deviceChanged) {}
207
208   private void buildGlPaper(GL gl, int index, Paper paper) {
209
210     // for phase 0 and 1
211     int i;
212     int j;
213     int numberOfEdges = paper.getNumberOfEdges();
214     int numberOfPolygon;
215
216     Polygon[] polygons;
217     Point[] polygonPoints;
218     Line line;
219
220
221
222     gl.glShadeModel(GL.GL_FLAT);
223     gl.glNormal3f(0.0f, 0.0f, 1.0f);
224
225     /* draw polygon */
226     numberOfPolygon = paper.getNumberOfPolygons();
227     polygons = paper.getPolygons();
228
229     for (i=0; i<numberOfPolygon; i++) {
230
231       polygonPoints = polygons[i].getPoints();
232
233       gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);
234
235       gl.glBegin(GL.GL_TRIANGLES);
236         for (j=0; j<3; j++){
237           gl.glVertex3d(polygonPoints[j].getXInReal() * ZOOM_MAGNIFICATION
238               , polygonPoints[j].getYInReal() * ZOOM_MAGNIFICATION
239               , polygonPoints[j].getZInReal() * ZOOM_MAGNIFICATION );
240
241         }
242
243
244       gl.glEnd();
245
246
247     }
248     /* draw lines */
249     for (i=0; i<numberOfEdges; i++) {
250       line = paper.getLine(i);
251
252         gl.glLineWidth(LINEWIDTH);
253
254
255       gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);
256
257       gl.glBegin(GL.GL_LINES);
258
259         gl.glVertex3f((float) line.getStartPoint().getXInReal() *
              ZOOM_MAGNIFICATION,
260             (float) line.getStartPoint().getYInReal() * ZOOM_MAGNIFICATION,
261             (float) line.getStartPoint().getZInReal() * ZOOM_MAGNIFICATION)
                  ;
262
263         gl.glVertex3f((float) line.getEndPoint().getXInReal() *
              ZOOM_MAGNIFICATION,
264             (float) line.getEndPoint().getYInReal() * ZOOM_MAGNIFICATION,
265             (float) line.getEndPoint().getZInReal() * ZOOM_MAGNIFICATION);
266
267       gl.glEnd();
268     }
269   }
270
271     // Methods required for the implementation of MouseListener
272   public void mouseEntered(MouseEvent e) {}
273   public void mouseExited(MouseEvent e) {}
274   public void mousePressed(MouseEvent e) {
275       prevMouseX = e.getX();
276       prevMouseY = e.getY();
277       if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
278         mouseRButtonDown = true;
279       }
280
281       if (mouseRButtonDown == true) {
282         isAnimating = true;
283       }
284
285
286   }
287
288   public void mouseReleased(MouseEvent e) {
289     if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
290       currentIndex_PaperGlId++;
291       currentIndex_PaperGlId = currentIndex_PaperGlId % numberOfPapers;
292       System.out.format("%d, %s\n", currentIndex_PaperGlId + 1, Integer.
              toString(currentIndex_PaperGlId + 1 ,2));
293       mouseRButtonDown = false;
294       }
295   }
296
297   public void mouseClicked(MouseEvent e) {}
298
299   // Methods required for the implementation of MouseMotionListener
300
301   public void mouseDragged(MouseEvent e) {
302     if (mouseRButtonDown == false) {
```

```
303
304         int x = e.getX();
305         int y = e.getY();
306         Dimension size = e.getComponent().getSize();
307
308         float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
309         float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);
310
311         prevMouseX = x;
312         prevMouseY = y;
313
314         view_rotx += thetaX;
315         view_roty += thetaY;
316       }else {
317
318       }
319     }
320
321     public void mouseMoved(MouseEvent e) {
322
323     }
324
325 }
```

```
1  package com.drancom.programmableMatter.folding.monitor;
2
3
4
5  import java.awt.Dimension;
6  import java.awt.Frame;
7  import java.awt.event.MouseEvent;
8  import java.awt.event.MouseListener;
9  import java.awt.event.MouseMotionListener;
10 import java.awt.event.WindowAdapter;
11 import java.awt.event.WindowEvent;
12
13 import javax.media.opengl.*;
14
15 import com.drancom.programmableMatter.folding.controller.paper.Paper;
16 import com.drancom.programmableMatter.folding.controller.paper.Point;
17 import com.drancom.programmableMatter.folding.controller.paper.Polygon;
18 import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
19 import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
       ;
20 import com.drancom.programmableMatter.folding.origami.planner.
       PlannerForWiring;
21 import com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamis.PlanForOrigamis;
22 import com.sun.opengl.util.Animator;
23
24
25 public class MonitorOfPlanGroupOfPlanForOrigamis  implements
       GLEventListener, MouseListener, MouseMotionListener {
26   final static float LINEWIDTH = 2.0f;
27   // final static float SIZE_OF_SQURE = 0.35f;
28   final static float SIZE_OF_SQURE = 0.35f;
29
30   // light
31   // float pos0[] =  {  -100.0f,  190.0f, 150.0f,  1.0f };
32   float pos0[] =  {  -100.0f,  100.0f, 100.0f,  1.0f };
33   float ambientLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
34   float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
35   float specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};
36
37   float white[] = {  1.0f,  1.0f,  1.0f,  0.2f };
38   float red[] =   {  1.0f,  0.0f,  0.0f,  1.0f };
39   float green[] = {  0.0f,  1.0f,  0.0f,  1.0f };
40   float yellow[] = {  1.0f,  1.0f,  0.0f,  1.0f };
41   float blue[] = { 0.0f, 0.0f, 1.0f, 0.0f };
```

```
42   float black[] = { 0.0f, 0.0f, 0.0f, 1.0f };
43
44   final static int SPEED_OF_ANIMATION = 15; //10 is default
45   int counterForSpeedOfAnimation = 0;
46
47   boolean isAnimating=false;
48
49
50   PlanForOrigamis planForOrigamis;
51   int    [][] planGroupTable;
52   int    [][][] planTable;
53   float  [][] edgeTable ;
54
55   int numberOfGroups;
56
57
58
59   public void run(PlanForOrigamis planForOrigamis) {
60     numberOfGroups = planForOrigamis.getNumberOfGroups();
61
62     paperGlId = new int[numberOfGroups];
63
64     this.planForOrigamis = planForOrigamis;
65     planTable = planForOrigamis.getPlanTable();
66     planGroupTable = planForOrigamis.getPlanGroupTable();
67     edgeTable = planForOrigamis.getEdgeTable();
68
69     Frame frame = new Frame("Monitor Of PaperArray - Programmable Matter by
           Folding - by Byoungkwon An");
70       GLCanvas canvas = new GLCanvas();
71
72     canvas.addGLEventListener(this);
73     frame.add(canvas);
74     frame.setSize(800, 800);
75     final Animator animator = new Animator(canvas);
76     frame.addWindowListener(new WindowAdapter() {
77       public void windowClosing(WindowEvent e) {
78           // Run this on another thread than the AWT event queue to
79         // make sure the call to Animator.stop() completes before
80         // exiting
81         new Thread(new Runnable() {
82           public void run() {
83             animator.stop();
84             System.exit(0);
85           }
86         }).start();
87           }
88     });
89       frame.show();
90       animator.start();
91   }
92
93   private float view_rotx = 0.0f, view_roty = 0.0f, view_rotz = 0.0f;
94
95   private int paperGlId[];
96   private int currentIndex_PaperGlId;
97
98   private boolean isFolding = true;
99
100  public boolean isFolding() {
101    return isFolding;
102  }
103
104  private int prevMouseX, prevMouseY;
105  private boolean mouseRButtonDown = false;
106
107  public void init(GLAutoDrawable drawable) {
108    int i;
109
```

```
110     // Use debug pipeline
111     // drawable.setGL(new DebugGL(drawable.getGL()));
112
113     GL gl = drawable.getGL();
114
115     System.err.println("INIT GL IS: " + gl.getClass().getName());
116     System.err.println("Chosen GLCapabilities: " + drawable.
            getChosenGLCapabilities());
117
118     gl.setSwapInterval(1);
119
120     // Blend
121     gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE);
122     gl.glEnable(GL.GL_BLEND);
123
124
125 //     gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
126 //       gl.glClearDepth(1.0f);
127
128     gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos0, 0);
129     gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambientLight, 0);
130     gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuseLight, 0);
131     gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular, 0);
132
133     gl.glEnable(GL.GL_CULL_FACE);
134
135     gl.glEnable(GL.GL_LIGHTING);
136       gl.glEnable(GL.GL_LIGHT0);
137
138 //     gl.glEnable(GL.GL_DEPTH_TEST);
139
140       /* make the papers */
141       for (i=0; i < numberOfGroups; i++){
142         paperGlId[i] = gl.glGenLists(1);
143         gl.glNewList(paperGlId[i], GL.GL_COMPILE);
144         buildGlPaper(gl, i, planForOrigamis);
145         gl.glEndList();
146       }
147
148     currentIndex_PaperGlId = 0;
149
150
151   //  gl.glEnable(GL.GL_NORMALIZE);
152
153       drawable.addMouseListener(this);
154       drawable.addMouseMotionListener(this);
155   }
156
157
158
159
160   public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
          height) {
161     GL gl = drawable.getGL();
162
163     float h = (float)height / (float)width;
164
165     gl.glMatrixMode(GL.GL_PROJECTION);
166
167     System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
168     System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
169     System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
170     gl.glLoadIdentity();
171     gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
172     gl.glMatrixMode(GL.GL_MODELVIEW);
173     gl.glLoadIdentity();
174     gl.glTranslatef(0.0f, 0.0f, -40.0f);
175   }
176
```

```
177   public void display(GLAutoDrawable drawable) {
178     // Turn the gears' teeth
179
180     // Get the GL corresponding to the drawable we are animating
181     GL gl = drawable.getGL();
182
183     // Special handling for the case where the GLJPanel is translucent
184     // and wants to be composited with other Java 2D content
185     if ((drawable instanceof GLJPanel) &&
186         !((GLJPanel) drawable).isOpaque() &&
187         ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
188       gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
189     } else {
190       gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
191     }
192
193
194     // Rotate the entire assembly of gears based on how the user
195     // dragged the mouse around
196     gl.glPushMatrix();
197     gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
198     gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
199     gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);
200
201     // Place the first gear and call its display list
202     gl.glPushMatrix();
203       gl.glTranslatef(-5.0f, -5.0f, 0.0f);
204       gl.glCallList(paperGlId[((int) currentIndex_PaperGlId)]);
205     gl.glPopMatrix();
206
207
208     // Remember that every push needs a pop; this one is paired with
209     // rotating the entire gear assembly
210     gl.glPopMatrix();
211
212   }
213
214   public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
          boolean deviceChanged) {}
215
216   private void buildGlPaper(GL gl, int groupNumber, PlanForOrigamis
          planForOrigamis) {
217
218     // for phase 0 and 1
219     int i;
220     int j;
221     int k;
222     int numberOfEdges = planForOrigamis.getNumberOfEdges();
223
224     float[][] edgeTable = planForOrigamis.getEdgeTable();
225     int   [][] planGroupTable = planForOrigamis.getPlanGroupTable();
226     float [][][][] vertexs = planForOrigamis.getVertexs();
227
228     gl.glShadeModel(GL.GL_FLAT);
229     gl.glNormal3f(0.0f, 0.0f, 1.0f);
230
231     /* draw lines */
232     for (i=0; i<numberOfEdges; i++) {
233
234         gl.glLineWidth(LINEWIDTH);
235
236       if(planGroupTable[groupNumber][i] == 1) {
237         // mount
238         gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, red, 0);
239       }else if (planGroupTable[groupNumber][i] == -1){
240         // valley
241         gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, blue, 0);
242
243       }else {
```

```
244           // just grid
245           gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);
246         }
247
248
249         gl.glBegin(GL.GL_LINES);
250
251         gl.glVertex3f((float)edgeTable[i][0]*8,
252             (float)edgeTable[i][1]*8,
253             (float)0.0*8);
254
255         gl.glVertex3f((float)edgeTable[i][2]*8,
256             (float)edgeTable[i][3]*8,
257             (float)0.0*8);
258         gl.glEnd();
259       }
260
261 /**/
262
263       for (i=0; i<9; i++) {
264         for (j=0; j<9; j++){
265           if(vertexs[groupNumber][i][j][3] == 1){
266             gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, red, 0);
267
268           } else
269           if(vertexs[groupNumber][i][j][3] == 2){
270             gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, white, 0);
271
272
273           }
274           if(vertexs[groupNumber][i][j][3] == 1 || vertexs[groupNumber][i][j
                ][3] == 2)
275           {
276             gl.glBegin(GL.GL_LINES);
277             gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 + (
                    SIZE_OF_SQURE / 2),
278                 (float)vertexs[groupNumber][i][j][1]*8 + (SIZE_OF_SQURE / 2),
279                 (float)0.0*8);
280             gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 + (
                    SIZE_OF_SQURE / 2),
281                 (float)vertexs[groupNumber][i][j][1]*8 - (SIZE_OF_SQURE / 2),
282                 (float)0.0*8);
283
284             gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 + (
                    SIZE_OF_SQURE / 2),
285                 (float)vertexs[groupNumber][i][j][1]*8 - (SIZE_OF_SQURE / 2),
286                 (float)0.0*8);
287             gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 - (
                    SIZE_OF_SQURE / 2),
288                 (float)vertexs[groupNumber][i][j][1]*8 - (SIZE_OF_SQURE / 2),
289                 (float)0.0*8);
290
291             gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 - (
                    SIZE_OF_SQURE / 2),
292                 (float)vertexs[groupNumber][i][j][1]*8 - (SIZE_OF_SQURE / 2),
293                 (float)0.0*8);
294             gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 - (
                    SIZE_OF_SQURE / 2),
295                 (float)vertexs[groupNumber][i][j][1]*8 + (SIZE_OF_SQURE / 2),
296                 (float)0.0*8);
297
298             gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 - (
                    SIZE_OF_SQURE / 2),
299                 (float)vertexs[groupNumber][i][j][1]*8 + (SIZE_OF_SQURE / 2),
300                 (float)0.0*8);
301             gl.glVertex3f((float)vertexs[groupNumber][i][j][0] *8 + (
                    SIZE_OF_SQURE / 2),
302                 (float)vertexs[groupNumber][i][j][1]*8 + (SIZE_OF_SQURE / 2),
303                 (float)0.0*8);
304
305             gl.glEnd();
306           }
307         }
308
309       }
310 /**/
311   }
312
313     // Methods required for the implementation of MouseListener
314   public void mouseEntered(MouseEvent e) {}
315   public void mouseExited(MouseEvent e) {}
316   public void mousePressed(MouseEvent e) {
317       prevMouseX = e.getX();
318       prevMouseY = e.getY();
319       if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
320         mouseRButtonDown = true;
321       }
322
323       if (mouseRButtonDown == true) {
324         isAnimating = true;
325       }
326
327
328   }
329
330   public void mouseReleased(MouseEvent e) {
331     if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
332       currentIndex_PaperGlId++;
333       currentIndex_PaperGlId = currentIndex_PaperGlId % numberOfGroups;
334       System.out.format("%d, %s\n", currentIndex_PaperGlId + 1, Integer.
              toString(currentIndex_PaperGlId + 1 ,2));
335       mouseRButtonDown = false;
336       }
337   }
338
339   public void mouseClicked(MouseEvent e) {}
340
341   // Methods required for the implementation of MouseMotionListener
342   public void mouseDragged(MouseEvent e) {
343     if (mouseRButtonDown == false) {
344
345       int x = e.getX();
346       int y = e.getY();
347       Dimension size = e.getComponent().getSize();
348
349       float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
350       float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);
351
352       prevMouseX = x;
353       prevMouseY = y;
354
355       view_rotx += thetaX;
356       view_roty += thetaY;
357     }else {
358
359     }
360   }
361
362   public void mouseMoved(MouseEvent e) {
363
364   }
365
366 }
```

```
1  package com.drancom.programmableMatter.folding.monitor;
2
3  import java.awt.*;
4  import java.awt.event.*;
```

```
5
6  import javax.media.opengl.*;
7
8  import com.drancom.programmableMatter.folding.controller.paper.Paper;
9  import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
10 import com.sun.opengl.util.*;
11
12 public class MonitorOfTileProgrammableMatter implements GLEventListener,
       MouseListener, MouseMotionListener {
13   Paper[] papers;
14
15   public void run(Paper[] papers) {
16     this.papers = papers;
17     paperGlId = new int[papers.length];
18
19     Frame frame = new Frame("Programmable Matter by Folding");
20       GLCanvas canvas = new GLCanvas();
21
22       canvas.addGLEventListener(this);
23       frame.add(canvas);
24       frame.setSize(800, 800);
25       final Animator animator = new Animator(canvas);
26       frame.addWindowListener(new WindowAdapter() {
27           public void windowClosing(WindowEvent e) {
28             // Run this on another thread than the AWT event queue to
29             // make sure the call to Animator.stop() completes before
30             // exiting
31             new Thread(new Runnable() {
32                 public void run() {
33                   animator.stop();
34                   System.exit(0);
35                 }
36               }).start();
37           }
38         });
39       frame.show();
40       animator.start();
41   }
42
43   private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;
44
45   private int paperGlId[];
46   private int currentIndex_PaperGlId;
47
48   private boolean isFolding = true;
49
50   public boolean isFolding() {
51     return isFolding;
52   }
53   private int prevMouseX, prevMouseY;
54   private boolean mouseRButtonDown = false;
55
56   public void init(GLAutoDrawable drawable) {
57     int i;
58
59     // Use debug pipeline
60     // drawable.setGL(new DebugGL(drawable.getGL()));
61
62     GL gl = drawable.getGL();
63
64     System.err.println("INIT GL IS: " + gl.getClass().getName());
65
66     System.err.println("Chosen GLCapabilities: " + drawable.
           getChosenGLCapabilities());
67
68     gl.setSwapInterval(1);
69
70     float pos[] = { 1.0f, 1.0f, -1.0f, 0.0f };
71     // float pos[] = { 0.0f, 0.0f, -1.0f, 0.0f };
72     float white[] = {1.0f, 1.0f, 1.0f, 1.0f};
73     float red[] = { 0.8f, 0.1f, 0.0f, 1.0f };
74     float green[] = { 0.0f, 0.8f, 0.2f, 1.0f };
75     float blue[] = { 0.2f, 0.2f, 1.0f, 1.0f };
76     float block[] = { 0.2f, 0.2f, 0.2f, 1.0f };
77
78     gl.glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
79
80     gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos, 0);
81     gl.glEnable(GL.GL_CULL_FACE);
82       gl.glEnable(GL.GL_LIGHTING);
83       gl.glEnable(GL.GL_LIGHT0);
84       gl.glEnable(GL.GL_DEPTH_TEST);
85
86       gl.glLineWidth(0.5f);
87
88       /* make the papers */
89       for (i=0; i<papers.length; i++){
90         paperGlId[i] = gl.glGenLists(1);
91         gl.glNewList(paperGlId[i], GL.GL_COMPILE);
92         gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT_AND_DIFFUSE,
               block, 0);
93         buildGlPaper(gl, papers[i]);
94         gl.glEndList();
95       }
96
97       currentIndex_PaperGlId = 0;
98
99
100
101      gl.glEnable(GL.GL_NORMALIZE);
102
103      drawable.addMouseListener(this);
104      drawable.addMouseMotionListener(this);
105  }
106
107  public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
         height) {
108    GL gl = drawable.getGL();
109
110    float h = (float)height / (float)width;
111
112    gl.glMatrixMode(GL.GL_PROJECTION);
113
114    System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
115    System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
116    System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
117    gl.glLoadIdentity();
118    gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
119    gl.glMatrixMode(GL.GL_MODELVIEW);
120    gl.glLoadIdentity();
121    gl.glTranslatef(0.0f, 0.0f, -40.0f);
122  }
123
124  public void display(GLAutoDrawable drawable) {
125    // Turn the gears' teeth
126
127    // Get the GL corresponding to the drawable we are animating
128    GL gl = drawable.getGL();
129
130    // Special handling for the case where the GLJPanel is translucent
131    // and wants to be composited with other Java 2D content
132    if ((drawable instanceof GLJPanel) &&
133        !((GLJPanel) drawable).isOpaque() &&
134        ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
135      gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
136    } else {
137      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
138    }
```

```
139
140
141     // Rotate the entire assembly of gears based on how the user
142     // dragged the mouse around
143     gl.glPushMatrix();
144     gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
145     gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
146     gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);
147
148     // Place the first gear and call its display list
149     gl.glPushMatrix();
150       gl.glTranslatef(-5.0f, -5.0f, 0.0f);
151       gl.glCallList(paperGlId[currentIndex_PaperGlId]);
152     gl.glPopMatrix();
153
154
155     // Remember that every push needs a pop; this one is paired with
156     // rotating the entire gear assembly
157     gl.glPopMatrix();
158     if (mouseRButtonDown == true){
159       if (isFolding() == true ){
160         currentIndex_PaperGlId--;
161
162         if (currentIndex_PaperGlId < 0) {
163           currentIndex_PaperGlId = 0;
164         }
165
166       } else {
167         currentIndex_PaperGlId++;
168
169         if (currentIndex_PaperGlId >= papers.length - 1 ) {
170           currentIndex_PaperGlId = papers.length - 1;
171         }
172       }
173     }
174   }
175
176   public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
          boolean deviceChanged) {}
177
178   public static void buildGlPaper(GL gl, Paper paper){
179     int i;
180     int numberOfLine = paper.getNumberOfEdges();
181     Vector startPointVector;
182     Vector endPointVector ;
183
184     gl.glShadeModel(GL.GL_FLAT);
185
186     gl.glNormal3f(0.0f, 0.0f, 1.0f);
187
188     /* draw lines */
189
190     gl.glBegin(GL.GL_LINES);
191
192     for (i=0; i<numberOfLine; i++) {
193
194       startPointVector = paper.getLine(i).getStartPoint().getVectorInReal()
              ;
195       endPointVector = paper.getLine(i).getEndPoint().getVectorInReal();
196
197       gl.glVertex3f((float)startPointVector.getX()*8,
198           (float)startPointVector.getY()*8,
199           (float)startPointVector.getZ()*8);
200
201       gl.glVertex3f((float)endPointVector.getX()*8,
202           (float)endPointVector.getY()*8,
203           (float)endPointVector.getZ()*8);
204     }
205     gl.glEnd();
206   }
207
208     // Methods required for the implementation of MouseListener
209   public void mouseEntered(MouseEvent e) {}
210   public void mouseExited(MouseEvent e) {}
211   public void mousePressed(MouseEvent e) {
212       prevMouseX = e.getX();
213       prevMouseY = e.getY();
214       if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
215         mouseRButtonDown = true;
216       }
217       if (mouseRButtonDown == true) {
218
219         if (currentIndex_PaperGlId <= 0) {
220           currentIndex_PaperGlId = 0;
221           isFolding = false;
222         } else if (currentIndex_PaperGlId >= papers.length - 1 ) {
223
224             currentIndex_PaperGlId = papers.length - 1;
225             isFolding = true;
226       }
227
228     }
229   }
230
231   public void mouseReleased(MouseEvent e) {
232     if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
233       mouseRButtonDown = false;
234       }
235
236   }
237
238   public void mouseClicked(MouseEvent e) {}
239
240   // Methods required for the implementation of MouseMotionListener
241   public void mouseDragged(MouseEvent e) {
242     if (mouseRButtonDown == false) {
243       int x = e.getX();
244       int y = e.getY();
245       Dimension size = e.getComponent().getSize();
246
247       float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
248       float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);
249
250       prevMouseX = x;
251       prevMouseY = y;
252
253       view_rotx += thetaX;
254       view_roty += thetaY;
255     }else {
256
257     }
258   }
259
260   public void mouseMoved(MouseEvent e) {
261
262   }
263 }
```

```
1  package com.drancom.programmableMatter.folding.monitor;
2
3  import java.awt.*;
4  import java.awt.event.*;
5
6  import javax.media.opengl.*;
7
8  import com.drancom.programmableMatter.folding.controller.paper.Paper;
9  import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
10 import com.sun.opengl.util.*;
```

```
11
12  public class PlayWindow implements GLEventListener, MouseListener,
        MouseMotionListener {
13    Paper[] papers;
14
15    public void run(Paper[] papers) {
16      this.papers = papers;
17      paperGlId = new int[papers.length];
18
19      Frame frame = new Frame("Programmable Matter by Folding");
20        GLCanvas canvas = new GLCanvas();
21
22        canvas.addGLEventListener(this);
23        frame.add(canvas);
24        frame.setSize(800, 800);
25        final Animator animator = new Animator(canvas);
26        frame.addWindowListener(new WindowAdapter() {
27            public void windowClosing(WindowEvent e) {
28              // Run this on another thread than the AWT event queue to
29              // make sure the call to Animator.stop() completes before
30              // exiting
31              new Thread(new Runnable() {
32                  public void run() {
33                    animator.stop();
34                    System.exit(0);
35                  }
36                }).start();
37            }
38          });
39        frame.show();
40        animator.start();
41    }
42
43    private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;
44
45    private int paperGlId[];
46    private int currentIndex_PaperGlId;
47
48    private boolean isFolding = true;
49
50    public boolean isFolding() {
51      return isFolding;
52    }
53    private int prevMouseX, prevMouseY;
54    private boolean mouseRButtonDown = false;
55
56    public void init(GLAutoDrawable drawable) {
57      int i;
58
59      // Use debug pipeline
60      // drawable.setGL(new DebugGL(drawable.getGL()));
61
62      GL gl = drawable.getGL();
63
64      System.err.println("INIT GL IS: " + gl.getClass().getName());
65
66      System.err.println("Chosen GLCapabilities: " + drawable.
            getChosenGLCapabilities());
67
68      gl.setSwapInterval(1);
69
70      float pos[] = { 1.0f, 1.0f, -1.0f, 0.0f };
71      // float pos[] = { 0.0f, 0.0f, -1.0f, 0.0f };
72      float white[] = {1.0f, 1.0f, 1.0f, 1.0f};
73      float red[] = { 0.8f, 0.1f, 0.0f, 1.0f };
74      float green[] = { 0.0f, 0.8f, 0.2f, 1.0f };
75      float blue[] = { 0.2f, 0.2f, 1.0f, 1.0f };
76      float block[] = { 0.2f, 0.2f, 0.2f, 1.0f };
77
78      gl.glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
79
80      gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos, 0);
81      gl.glEnable(GL.GL_CULL_FACE);
82        gl.glEnable(GL.GL_LIGHTING);
83        gl.glEnable(GL.GL_LIGHT0);
84        gl.glEnable(GL.GL_DEPTH_TEST);
85
86        gl.glLineWidth(0.5f);
87
88        /* make the papers */
89        for (i=0; i<papers.length; i++){
90          paperGlId[i] = gl.glGenLists(1);
91          gl.glNewList(paperGlId[i], GL.GL_COMPILE);
92          gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT_AND_DIFFUSE,
                block, 0);
93          buildGlPaper(gl, papers[i]);
94          gl.glEndList();
95        }
96
97        currentIndex_PaperGlId = 0;
98
99
100
101       gl.glEnable(GL.GL_NORMALIZE);
102
103       drawable.addMouseListener(this);
104       drawable.addMouseMotionListener(this);
105   }
106
107   public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
          height) {
108     GL gl = drawable.getGL();
109
110     float h = (float)height / (float)width;
111
112     gl.glMatrixMode(GL.GL_PROJECTION);
113
114     System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
115     System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
116     System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
117     gl.glLoadIdentity();
118     gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
119     gl.glMatrixMode(GL.GL_MODELVIEW);
120     gl.glLoadIdentity();
121     gl.glTranslatef(0.0f, 0.0f, -40.0f);
122   }
123
124   public void display(GLAutoDrawable drawable) {
125     // Turn the gears' teeth
126
127     // Get the GL corresponding to the drawable we are animating
128     GL gl = drawable.getGL();
129
130     // Special handling for the case where the GLJPanel is translucent
131     // and wants to be composited with other Java 2D content
132     if ((drawable instanceof GLJPanel) &&
133         !((GLJPanel) drawable).isOpaque() &&
134         ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
135       gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
136     } else {
137       gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
138     }
139
140
141     // Rotate the entire assembly of gears based on how the user
142     // dragged the mouse around
143     gl.glPushMatrix();
144     gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
```

```
145     gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
146     gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);
147
148     // Place the first gear and call its display list
149     gl.glPushMatrix();
150       gl.glTranslatef(-5.0f, -5.0f, 0.0f);
151       gl.glCallList(paperGlId[currentIndex_PaperGlId]);
152     gl.glPopMatrix();
153
154
155     // Remember that every push needs a pop; this one is paired with
156     // rotating the entire gear assembly
157     gl.glPopMatrix();
158     if (mouseRButtonDown == true){
159       if (isFolding() == true ){
160         currentIndex_PaperGlId--;
161
162         if (currentIndex_PaperGlId < 0) {
163           currentIndex_PaperGlId = 0;
164         }
165
166       } else {
167         currentIndex_PaperGlId++;
168
169         if (currentIndex_PaperGlId >= papers.length - 1 ) {
170           currentIndex_PaperGlId = papers.length - 1;
171         }
172       }
173     }
174   }
175
176   public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
          boolean deviceChanged) {}
177
178   public static void buildGlPaper(GL gl, Paper paper){
179     int i;
180     int numberOfLine = paper.getNumberOfEdges();
181     Vector startPointVector;
182     Vector endPointVector ;
183
184     gl.glShadeModel(GL.GL_FLAT);
185
186     gl.glNormal3f(0.0f, 0.0f, 1.0f);
187
188     /* draw lines */
189
190     gl.glBegin(GL.GL_LINES);
191
192     for (i=0; i<numberOfLine; i++) {
193
194       startPointVector = paper.getLine(i).getStartPoint().getVectorInReal()
              ;
195       endPointVector = paper.getLine(i).getEndPoint().getVectorInReal();
196
197       gl.glVertex3f((float)startPointVector.getX()*8,
198           (float)startPointVector.getY()*8,
199           (float)startPointVector.getZ()*8);
200
201       gl.glVertex3f((float)endPointVector.getX()*8,
202           (float)endPointVector.getY()*8,
203           (float)endPointVector.getZ()*8);
204     }
205     gl.glEnd();
206   }
207
208     // Methods required for the implementation of MouseListener
209   public void mouseEntered(MouseEvent e) {}
210   public void mouseExited(MouseEvent e) {}
211   public void mousePressed(MouseEvent e) {
212       prevMouseX = e.getX();
213       prevMouseY = e.getY();
214       if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
215         mouseRButtonDown = true;
216       }
217       if (mouseRButtonDown == true) {
218
219         if (currentIndex_PaperGlId <= 0) {
220           currentIndex_PaperGlId = 0;
221           isFolding = false;
222         } else if (currentIndex_PaperGlId >= papers.length - 1 ) {
223
224           currentIndex_PaperGlId = papers.length - 1;
225           isFolding = true;
226       }
227
228     }
229   }
230
231   public void mouseReleased(MouseEvent e) {
232     if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
233       mouseRButtonDown = false;
234       }
235
236   }
237
238   public void mouseClicked(MouseEvent e) {}
239
240   // Methods required for the implementation of MouseMotionListener
241   public void mouseDragged(MouseEvent e) {
242     if (mouseRButtonDown == false) {
243       int x = e.getX();
244       int y = e.getY();
245       Dimension size = e.getComponent().getSize();
246
247       float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
248       float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);
249
250       prevMouseX = x;
251       prevMouseY = y;
252
253       view_rotx += thetaX;
254       view_roty += thetaY;
255     }else {
256
257     }
258   }
259
260   public void mouseMoved(MouseEvent e) {
261
262   }
263 }
```

```
1  package com.drancom.programmableMatter.folding.monitor;
2
3  import java.awt.*;
4  import java.awt.event.*;
5
6  import javax.media.opengl.*;
7
8  import com.drancom.programmableMatter.folding.controller.paper.Paper;
9  import com.drancom.programmableMatter.folding.controller.paper.
       UnfoldingPaper;
10 import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
11 import com.sun.opengl.util.*;
12
13 public class UnfoldingWindow extends MainWindow implements GLEventListener,
       MouseListener, MouseMotionListener {
14   Paper paper;
```

```
15
16    GLCanvas canvas;
17
18    final static float pos[] = { 1.0f, 1.0f, -1.0f, 0.0f };
19    // float pos[] = { 0.0f, 0.0f, -1.0f, 0.0f };
20    final static float white[] = {1.0f, 1.0f, 1.0f, 1.0f};
21    final static float red[] = { 0.8f, 0.1f, 0.0f, 1.0f };
22    final static float green[] = { 0.0f, 0.8f, 0.2f, 1.0f };
23    final static float blue[] = { 0.2f, 0.2f, 1.0f, 1.0f };
24    final static float block[] = { 0.2f, 0.2f, 0.2f, 1.0f };
25
26
27    private float view_rotx = 20.0f, view_roty = 30.0f, view_rotz = 0.0f;
28
29    private int paperGlId;
30    private int currentIndex_PaperGlId;
31
32    private boolean isFolding = true;
33
34
35    private int prevMouseX, prevMouseY;
36    private boolean mouseRButtonDown = false;
37
38    public void run(Paper paper) {
39      this.paper = paper;
40
41      Frame frame = new Frame("Programmable Matter by Folding");
42        canvas = new GLCanvas();
43
44        canvas.addGLEventListener(this);
45        frame.add(canvas);
46        frame.setSize(800, 800);
47        final Animator animator = new Animator(canvas);
48        frame.addWindowListener(new WindowAdapter() {
49            public void windowClosing(WindowEvent e) {
50              // Run this on another thread than the AWT event queue to
51              // make sure the call to Animator.stop() completes before
52              // exiting
53              new Thread(new Runnable() {
54                  public void run() {
55                    animator.stop();
56                    System.exit(0);
57                  }
58                }).start();
59            }
60          });
61        frame.show();
62        animator.start();
63    }
64
65    public void init(GLAutoDrawable drawable) {
66      int i;
67
68      // Use debug pipeline
69      // drawable.setGL(new DebugGL(drawable.getGL()));
70
71      GL gl = drawable.getGL();
72
73      System.err.println("INIT GL IS: " + gl.getClass().getName());
74
75      System.err.println("Chosen GLCapabilities: " + drawable.
            getChosenGLCapabilities());
76
77      gl.setSwapInterval(1);
78
79
80      gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos, 0);
81      gl.glEnable(GL.GL_CULL_FACE);
82        gl.glEnable(GL.GL_LIGHTING);
83        gl.glEnable(GL.GL_LIGHT0);
84        gl.glEnable(GL.GL_DEPTH_TEST);
85
86        gl.glLineWidth(0.5f);
87    /*
88        for (i=0; i<papers.length; i++){
89          paperGlId[i] = gl.glGenLists(1);
90          gl.glNewList(paperGlId[i], GL.GL_COMPILE);
91          gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT_AND_DIFFUSE,
                white, 0);
92          buildGlPaper(gl, papers[i]);
93          gl.glEndList();
94        }
95    */
96        currentIndex_PaperGlId = 0;
97
98        gl.glEnable(GL.GL_NORMALIZE);
99
100       drawable.addMouseListener(this);
101       drawable.addMouseMotionListener(this);
102   }
103
104   public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
          height) {
105     GL gl = drawable.getGL();
106
107     float h = (float)height / (float)width;
108
109     gl.glMatrixMode(GL.GL_PROJECTION);
110
111     System.err.println("GL_VENDOR: " + gl.glGetString(GL.GL_VENDOR));
112     System.err.println("GL_RENDERER: " + gl.glGetString(GL.GL_RENDERER));
113     System.err.println("GL_VERSION: " + gl.glGetString(GL.GL_VERSION));
114     gl.glLoadIdentity();
115     gl.glFrustum(-1.0f, 1.0f, -h, h, 5.0f, 60.0f);
116     gl.glMatrixMode(GL.GL_MODELVIEW);
117     gl.glLoadIdentity();
118     gl.glTranslatef(0.0f, 0.0f, -40.0f);
119   }
120
121   public void display(GLAutoDrawable drawable) {
122     // Turn the gears' teeth
123
124     // Get the GL corresponding to the drawable we are animating
125     GL gl = drawable.getGL();
126
127     // Special handling for the case where the GLJPanel is translucent
128     // and wants to be composited with other Java 2D content
129     if ((drawable instanceof GLJPanel) &&
130         !((GLJPanel) drawable).isOpaque() &&
131         ((GLJPanel) drawable).shouldPreserveColorBufferIfTranslucent()) {
132       gl.glClear(GL.GL_DEPTH_BUFFER_BIT);
133     } else {
134       gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
135     }
136
137
138     // Rotate the entire assembly of gears based on how the user
139     // dragged the mouse around
140     gl.glPushMatrix();
141     gl.glRotatef(view_rotx, 1.0f, 0.0f, 0.0f);
142     gl.glRotatef(view_roty, 0.0f, 1.0f, 0.0f);
143     gl.glRotatef(view_rotz, 0.0f, 0.0f, 1.0f);
144
145     // Place the first gear and call its display list
146     gl.glPushMatrix();
147       gl.glTranslatef(-5.0f, -5.0f, 0.0f);
148       //gl.glCallList(paperGlId[currentIndex_PaperGlId]);
149         gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT_AND_DIFFUSE,
```

```
                white, 0);
150           buildGlPaper(gl, paper);
151
152       gl.glPopMatrix();
153
154
155       // Remember that every push needs a pop; this one is paired with
156       // rotating the entire gear assembly
157       gl.glPopMatrix();
158     }
159
160     public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
          boolean deviceChanged) {}
161
162     public boolean isFolding() {
163       return isFolding;
164     }
165
166     public void buildGlPaper(GL gl, Paper paper){
167       int i;
168       int numberOfLine = paper.getNumberOfEdges();
169       Vector startPointVector;
170       Vector endPointVector ;
171
172       gl.glShadeModel(GL.GL_FLAT);
173
174       gl.glNormal3f(0.0f, 0.0f, 1.0f);
175
176       /* draw lines */
177
178       gl.glBegin(GL.GL_LINES);
179
180       for (i=0; i<numberOfLine; i++) {
181
182         startPointVector = paper.getLine(i).getStartPoint().getVectorInReal()
                ;
183         endPointVector = paper.getLine(i).getEndPoint().getVectorInReal();
184
185         gl.glVertex3f((float)startPointVector.getX()*8,
186             (float)startPointVector.getY()*8,
187             (float)startPointVector.getZ()*8);
188
189         gl.glVertex3f((float)endPointVector.getX()*8,
190             (float)endPointVector.getY()*8,
191             (float)endPointVector.getZ()*8);
192       }
193       gl.glEnd();
194     }
195
196       // Methods required for the implementation of MouseListener
197     public void mouseEntered(MouseEvent e) {}
198     public void mouseExited(MouseEvent e) {}
199     public void mousePressed(MouseEvent e) {
200         prevMouseX = e.getX();
201         prevMouseY = e.getY();
202         if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
203           mouseRButtonDown = true;
204         }
205         if (mouseRButtonDown == true) {
206 //            animator.stop();
207
208             final Animator animator = new Animator(canvas);
209
210           animator.stop();
211           Paper snapshotPaper = paper.snapshot();
212
213 //          ((UnfoldingPaper) snapshotPaper).unfolding(0.1f);
214
215           paper = snapshotPaper;
```

```
216           animator.start();
217
218       }
219     }
220
221     public void mouseReleased(MouseEvent e) {
222       if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
223         mouseRButtonDown = false;
224
225         }
226
227     }
228
229     public void mouseClicked(MouseEvent e) {}
230
231     // Methods required for the implementation of MouseMotionListener
232     public void mouseDragged(MouseEvent e) {
233       if (mouseRButtonDown == false) {
234         int x = e.getX();
235         int y = e.getY();
236         Dimension size = e.getComponent().getSize();
237
238         float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
239         float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);
240
241         prevMouseX = x;
242         prevMouseY = y;
243
244         view_rotx += thetaX;
245         view_roty += thetaY;
246       }else {
247
248       }
249     }
250
251     public void mouseMoved(MouseEvent e) {
252
253     }
254 }
```

```
1  package com.drancom.programmableMatter.folding.origami.planner;
2
3  public interface Plan {
4
5  }
```

```
1  package com.drancom.programmableMatter.folding.origami.planner;
2
3  public class PlanForAngleActurator implements Plan {
4
5  }
```

```
1  package com.drancom.programmableMatter.folding.origami.planner;
2
3  public class PlanForWiring implements Plan {
4    // folding or unfolding []
5    // phases []
6    // inside or outside[]
7    // numberOfEdge []
8    // data true or false
9    boolean planTable[][][][];
10   int numberOfPhases;
11   int numberOfEdges;
12
13   PlanForWiring (int maxPhases, int numberOfEdges) {
14     int i;
15     int j;
```

```
16      int k;
17      int l;
18
19      numberOfPhases = 0;
20      this.numberOfEdges = numberOfEdges;
21      planTable = new boolean[2][maxPhases][2][numberOfEdges];
22
23      for (i=0; i<2; i++) {
24        for (j=0; j<maxPhases; j++) {
25          for(k=0; k<2 ; k++) {
26            for(l=0; l<numberOfEdges; l++) {
27              planTable[i][j][k][l] = false;
28            }
29          }
30        }
31      }
32    }
33
34    public boolean setPlanForWiring(int folding
35        , int phase
36        , int inside
37        , int edgeNumber
38        , boolean active) {
39
40      planTable[folding][phase][inside][edgeNumber] = active;
41
42      return true;
43    }
44
45    public void setNumberOfPhases(int numberOfPhases) {
46      this.numberOfPhases = numberOfPhases;
47    }
48
49    public boolean [][][][][] getPlanTable() {
50      return planTable;
51    }
52
53    public int getNumberOfPhases() {
54      return numberOfPhases;
55    }
56
57    public int getNumberOfEdges() {
58      return numberOfEdges;
59    }
60  }
```

```
1  package com.drancom.programmableMatter.folding.origami.planner;
2
3  import com.drancom.programmableMatter.folding.controller.paper.Paper;
4
5  public interface Planner {
6
7    public void build(Paper[] papers);
8    public void build(Plan[] plans);
9
10   public Plan getPlan();
11   public void exportPlan(String fileName);
12   public void exportPlan(String fileName, Paper[] papers);
13 }
```

```
1  package com.drancom.programmableMatter.folding.origami.planner;
2
3  import com.drancom.programmableMatter.folding.controller.paper.Paper;
4  import com.drancom.programmableMatter.folding.dataFile.FileAngleData;
5  import com.drancom.programmableMatter.folding.dataFile.FilePlan;
6  import com.drancom.programmableMatter.folding.dataFile.
       FilePlanForAngleActuration;
7
```

```
8  public class PlannerForAngleActurator implements Planner {
9
10   public void build(Paper[] papers) {
11
12     int i;
13     int j;
14
15     int numberOfLines;
16     float levelOfActuratingPower;
17
18     numberOfLines = papers[0].getNumberOfEdges();
19
20     for (i=0; i<papers.length -1 ; i++) {
21       for(j=0 ; j<numberOfLines ; j++) {
22         levelOfActuratingPower = papers[i].getLine(j).getAngle() - papers[i
               +1].getLine(j).getAngle();
23
24         if (levelOfActuratingPower > 0.15f) {
25           papers[i+1].getLine(j).setLevelOfActuratingPower( 2.0f);
26         } else if (levelOfActuratingPower < -0.15) {
27           papers[i+1].getLine(j).setLevelOfActuratingPower(-2.0f);
28         } else if (levelOfActuratingPower > 0.0) {
29           papers[i+1].getLine(j).setLevelOfActuratingPower( 1.0f);
30         } else if (levelOfActuratingPower < 0.0) {
31           papers[i+1].getLine(j).setLevelOfActuratingPower(-1.0f);
32         } else {
33           papers[i+1].getLine(j).setLevelOfActuratingPower(0.0f);
34         }
35       }
36     }
37
38     for(j=0 ; j<numberOfLines ; j++) {
39
40       papers[0].getLine(j).setLevelOfActuratingPower(0.0f);
41
42     }
43   }
44
45   public void exportPlan(String fileName, Paper[] papers) {
46     FilePlan filePlan = new FilePlanForAngleActuration();
47
48     filePlan.build(fileName, papers);
49   }
50
51   @Override
52   public Plan getPlan() {
53     return null;
54   }
55
56   @Override
57   public void build(Plan[] plans) {
58     // TODO Auto-generated method stub
59
60   }
61
62   @Override
63   public void exportPlan(String fileName) {
64     // TODO Auto-generated method stub
65
66   }
67
68 }
```

```
1  package com.drancom.programmableMatter.folding.origami.planner;
2
3  import com.drancom.programmableMatter.folding.controller.paper.Paper;
4  import com.drancom.programmableMatter.folding.dataFile.FilePlan;
5  import com.drancom.programmableMatter.folding.dataFile.FilePlanForWiring;
6
```

```
7
8  public class PlannerForWiring implements Planner {
9
10   final int MAX_PHEASES = 10;
11   final float STANDARD_NUMBER_FOR_ACTIVATION = 0.0f;
12   final int STANDARD_NUMBER_OF_PHASE_COUNTER_FOR_PHASE_CHANGE = 4;
13
14
15   PlanForWiring planForWiring;
16
17   public void build(Paper[] papers) {
18
19     int i;
20     int j;
21
22     int numberOfLines;
23     float levelOfActuratingPower;
24
25     int phase = 0;
26     int phaseCounter = 0;
27
28     float standard_number_for_Activation = STANDARD_NUMBER_FOR_ACTIVATION;
29
30     numberOfLines = papers[0].getNumberOfEdges();
31     planForWiring = new PlanForWiring(10, numberOfLines);
32
33
34
35     for (i=0; i<papers.length -1 ; i++) {
36       for(j=0 ; j<numberOfLines ; j++) {
37         levelOfActuratingPower = papers[i].getLine(j).getAngle() - papers[i
               +1].getLine(j).getAngle();
38
39         if (levelOfActuratingPower > standard_number_for_Activation) {
40           if (phaseCounter >=
                  STANDARD_NUMBER_OF_PHASE_COUNTER_FOR_PHASE_CHANGE *
                  numberOfLines) {
41             phase++;
42             planForWiring.setNumberOfPhases(phase);
43           }
44
45           planForWiring.setPlanForWiring(0, phase, 0, j, true);
46                 phaseCounter = 0;
47
48         } else if (levelOfActuratingPower < -1 *
                standard_number_for_Activation ) {
49           if (phaseCounter >=
                  STANDARD_NUMBER_OF_PHASE_COUNTER_FOR_PHASE_CHANGE *
                  numberOfLines) {
50             phase++;
51             planForWiring.setNumberOfPhases(phase);
52           }
53
54           planForWiring.setPlanForWiring(0, phase, 1, j, true);
55                 phaseCounter = 0;
56         } else {
57           phaseCounter++;
58         }
59       }
60     }
61   }
62
63   public void exportPlan(String fileName, Paper[] papers) {
64     FilePlan filePlan = new FilePlanForWiring();
65
66     filePlan.build(fileName, papers);
67   }
68
69   public Plan getPlan() {
70
71     return (Plan) planForWiring;
72   }
73
74   @Override
75   public void build(Plan[] plans) {
76     // TODO Auto-generated method stub
77
78   }
79
80   @Override
81   public void exportPlan(String fileName) {
82     // TODO Auto-generated method stub
83
84   }
85 }
```

```
1  package com.drancom.programmableMatter.folding.simulator;
2
3  import com.drancom.programmableMatter.folding.controller.paper.Paper;
4  import com.drancom.programmableMatter.folding.dataFile.FileAngleData;
5  import com.drancom.programmableMatter.folding.dataFile.FileObj;
6  import com.drancom.programmableMatter.folding.monitor.MainWindow;
7
8  public class AngleDataCollector {
9
10 /** /
11     public static final String FILENAME = "c:\\foldingdata\\save_airplain\\m
           %05d.obj";
12     public static final String FILEANGLENAME = "c:\\foldingdata\\
           save_airplain\\angle.csv";
13     public static final int NUMBER_OF_FILES= 50;
14 /** /
15   public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d.
         obj";
16   public static final String FILEANGLENAME = "c:\\foldingdata\\save_box\\
         angle.csv";
17   public static final int NUMBER_OF_FILES= 70;
18 /** /
19   public static final String FILENAME = "c:\\foldingdata\\save_cup\\m%05d.
         obj";
20   public static final String FILEANGLENAME = "c:\\foldingdata\\save_cup\\
         angle.csv";
21   public static final int NUMBER_OF_FILES= 140;
22 /**/
23   public static final String PLAN_FILENAME = "c:\\foldingdata\\save_bench\\
         plan_save_bench.csv";
24   public static final String FILENAME = "c:\\foldingdata\\save_bench\\m%05d
         .obj";
25   public static final String FILEANGLENAME = "c:\\foldingdata\\save_bench\\
         angle.csv";
26   public static final int NUMBER_OF_FILES= 70;
27 /**/
28 //  public static final String FILENAME = "c:\\foldingdata\\save_box\\
        m00070.obj";
29 //  public static final int NUMBER_OF_FILES= 1;
30
31   void run() {
32
33     int i;
34     String fileName;
35     String fileAngleName;
36
37     Paper[] papers;
38     FileObj[] fileObjs;
39     FileAngleData fileAngleData;
40
41     // init
42     papers = new Paper[NUMBER_OF_FILES];
```

```
43      fileObjs = new FileObj[NUMBER_OF_FILES];
44      fileAngleData = new FileAngleData();
45
46      // load
47      for (i=0; i < NUMBER_OF_FILES; i++) {
48
49        fileName = String.format(FILENAME, i+1);
50        papers[i] = new Paper();
51        fileObjs[i] = new FileObj();
52        fileObjs[i].load(fileName, papers[i]);
53      }
54
55      // save
56      fileAngleName = String.format(FILEANGLENAME);
57      fileAngleData.build(fileAngleName, papers);
58    }
59
60    public static void main(String[] args) {
61      // TODO Auto-generated method stub
62      AngleDataCollector simulator = new AngleDataCollector();
63      simulator.run();
64
65    }
66  }
```

```
1   package com.drancom.programmableMatter.folding.simulator;
2
3   import com.drancom.programmableMatter.folding.controller.paper.Paper;
4   import com.drancom.programmableMatter.folding.dataFile.FileObj;
5   import com.drancom.programmableMatter.folding.monitor.MainWindow;
6
7   public class Player {
8   /**/
9
10    public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
          RigidOrigami006\\RigidOrigami\\save_4x4_s-shuttle\\m%05d.obj";
11    public static final int NUMBER_OF_FILES= 40;
12  /** /
13    public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
          RigidOrigami006\\RigidOrigami\\save_4x4_pyramid\\m%05d.obj";
14    public static final int NUMBER_OF_FILES= 40;
15  /** /
16    public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
          RigidOrigami006\\RigidOrigami\\save_8x8_s-shuttle\\m%05d.obj";
17    public static final int NUMBER_OF_FILES= 50;
18  /** /
19    public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
          RigidOrigami006\\RigidOrigami\\save_8x8_hat\\m%05d.obj";
20    public static final int NUMBER_OF_FILES= 28;
21  /** /
22    public static final String FILENAME = "c:\\foldingdata\\save_airplain\\m
          %05d.obj";
23    public static final int NUMBER_OF_FILES= 70;
24  /** /
25    public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d.
          obj";
26    public static final int NUMBER_OF_FILES= 70;
27  /**/
28  //  public static final String FILENAME = "c:\\foldingdata\\save_box\\
        m00070.obj";
29  //  public static final int NUMBER_OF_FILES= 1;
30
31    Paper[] papers;
32    FileObj[] fileObjs;
33    MainWindow mainWindow;
34    public Player() {
35
36    }
37
```

```
38    void run() {
39      int i;
40      String fileName;
41
42      // init
43      papers = new Paper[NUMBER_OF_FILES];
44      fileObjs = new FileObj[NUMBER_OF_FILES];
45      mainWindow = new MainWindow();
46
47      // load
48      for (i=0; i < NUMBER_OF_FILES; i++) {
49
50        fileName = String.format(FILENAME, i+1);
51        papers[i] = new Paper();
52        fileObjs[i] = new FileObj();
53        fileObjs[i].load(fileName, papers[i]);
54
55      }
56
57      mainWindow.run(papers);
58
59    }
60
61
62    public static void main(String[] args) {
63      // TODO Auto-generated method stub
64      Player simulator = new Player();
65      simulator.run();
66    }
67  }
```

```
1   package com.drancom.programmableMatter.folding.simulator;
2
3   import com.drancom.programmableMatter.folding.controller.paper.Paper;
4   import com.drancom.programmableMatter.folding.dataFile.FileObj;
5   import com.drancom.programmableMatter.folding.monitor.MainWindow;
6   import com.drancom.programmableMatter.folding.origami.planner.Planner;
7   import com.drancom.programmableMatter.folding.origami.planner.
        PlannerForWiring;
8
9   public class Simulator {
10
11  /** /
12    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_airplain\\plan_airplain.csv";
13    public static final String FILENAME = "c:\\foldingdata\\save_airplain\\m
          %05d.obj";
14    public static final int NUMBER_OF_FILES= 50;
15
16  /** /
17    public static final String PLAN_FILENAME = "c:\\foldingdata\\save_box\\
          plan_box.csv";
18    public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d.
          obj";
19    public static final int NUMBER_OF_FILES= 70;
20
21  /** /
22    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_sailboat2\\plan_sailboat2.csv";
23    public static final String FILENAME = "c:\\foldingdata\\save_sailboat2\\m
          %05d.obj";
24    public static final int NUMBER_OF_FILES= 95;
25  /**/
26    public static final String PLAN_FILENAME = "c:\\foldingdata\\
          save_8x8bench\\plan_save_bench.csv";
27    public static final String FILENAME = "c:\\foldingdata\\save_8x8bench\\m
          %05d.obj";
28    public static final int NUMBER_OF_FILES= 70;
29  /**/
```

```
30
31 //   public static final String FILENAME = "c:\\foldingdata\\save_box\\
       m00070.obj";
32 //   public static final int NUMBER_OF_FILES= 1;
33
34   Paper[] papers;
35   FileObj[] fileObjs;
36   MainWindow mainWindow;
37   public Simulator() {
38   }
39
40   void run() {
41     int i;
42     String fileName;
43
44     // init
45     papers = new Paper[NUMBER_OF_FILES];
46     fileObjs = new FileObj[NUMBER_OF_FILES];
47     mainWindow = new MainWindow();
48
49     // load
50     for (i=0; i < NUMBER_OF_FILES; i++) {
51       fileName = String.format(FILENAME, i+1);
52       papers[i] = new Paper();
53       fileObjs[i] = new FileObj();
54       fileObjs[i].load(fileName, papers[i]);
55     }
56
57     Planner planer = new PlannerForWiring();
58
59     planer.build(papers);
60
61     planer.exportPlan(PLAN_FILENAME, papers);
62
63     mainWindow.run(papers);
64
65   }
66
67   public static void main(String[] args) {
68     // TODO Auto-generated method stub
69     Simulator simulator = new Simulator();
70     simulator.run();
71   }
72 }
```

```
1  package com.drancom.programmableMatter.folding.simulator;
2
3  import com.drancom.programmableMatter.folding.controller.paper.Paper;
4  import com.drancom.programmableMatter.folding.dataFile.FileObj;
5  import com.drancom.programmableMatter.folding.monitor.MainWindow;
6  import com.drancom.programmableMatter.folding.monitor.
       MainWindowForFoldingRobotWiring;
7  import com.drancom.programmableMatter.folding.origami.planner.PlanForWiring
       ;
8  import com.drancom.programmableMatter.folding.origami.planner.Planner;
9  import com.drancom.programmableMatter.folding.origami.planner.
       PlannerForWiring;
10
11 public class SimulatorForFoildingWithWire {
12
13   /** /
14     public static final String PLAN_FILENAME = "c:\\foldingdata\\
           save_airplain\\plan_for_wiring_airplain.csv";
15     public static final String FILENAME = "c:\\foldingdata\\save_airplain\\
           m%05d.obj";
16     public static final int NUMBER_OF_FILES= 50;
17
18   /** /
```

```
19     public static final String PLAN_FILENAME = "c:\\foldingdata\\save_box\\
           plan_for_wiring_box.csv";
20     public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d
           .obj";
21     public static final int NUMBER_OF_FILES= 70;
22
23   /** /
24     public static final String PLAN_FILENAME = "c:\\foldingdata\\
           save_sailboat2\\plan_for_wiring_sailboat2.csv";
25     public static final String FILENAME = "c:\\foldingdata\\save_sailboat2
           \\m%05d.obj";
26     public static final int NUMBER_OF_FILES= 35;
27
28   /**/
29     public static final String PLAN_FILENAME = "c:\\foldingdata\\save_bench
           \\plan_for_wiring_save_bench.csv";
30     public static final String FILENAME = "c:\\foldingdata\\save_bench\\m
           %05d.obj";
31     public static final int NUMBER_OF_FILES= 70;
32
33   /**/
34
35 //     public static final String FILENAME = "c:\\foldingdata\\save_box\\
       m00070.obj";
36 //     public static final int NUMBER_OF_FILES= 1;
37
38     Paper[] papers;
39     FileObj[] fileObjs;
40     MainWindowForFoldingRobotWiring mainWindow;
41     public SimulatorForFoildingWithWire() {
42
43     }
44
45     void run() {
46       int i;
47       String fileName;
48
49       // Initiation
50       papers = new Paper[NUMBER_OF_FILES];
51       fileObjs = new FileObj[NUMBER_OF_FILES];
52        mainWindow = new MainWindowForFoldingRobotWiring();
53
54       // load
55       for (i=0; i < NUMBER_OF_FILES; i++) {
56         fileName = String.format(FILENAME, i+1);
57         papers[i] = new Paper();
58         fileObjs[i] = new FileObj();
59         fileObjs[i].load(fileName, papers[i]);
60       }
61
62       Planner planer = new PlannerForWiring(); // new planer
63
64       planer.build(papers);
65
66       planer.exportPlan(PLAN_FILENAME, papers);
67
68       mainWindow.run(papers, (PlanForWiring) planer.getPlan());
69
70     }
71
72     public static void main(String[] args) {
73       // TODO Auto-generated method stub
74       SimulatorForFoildingWithWire simulator = new
             SimulatorForFoildingWithWire ();
75       simulator.run();
76     }
77 }
```

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

```
2
3  import com.drancom.programmableMatter.folding.controller.paper.Paper;
4  import com.drancom.programmableMatter.folding.controller.paper.
       UnfoldingPaper;
5  import com.drancom.programmableMatter.folding.dataFile.FileObj;
6  import com.drancom.programmableMatter.folding.monitor.MainWindow;
7  import com.drancom.programmableMatter.folding.monitor.UnfoldingWindow;
8
9  public class UnfoldingSimulator {
10 /**/
11   public static final String FILENAME = "c:\\foldingdata\\save_airplain\\m
       %05d.obj";
12   public static final int NUMBER_OF_FILES= 81;
13 /** /
14   public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d.
       obj";
15   public static final int NUMBER_OF_FILES= 70;
16 /**/
17 //   public static final String FILENAME = "c:\\foldingdata\\save_box\\
       m00070.obj";
18 //   public static final int NUMBER_OF_FILES= 1;
19 /**/
20
21   Paper paper;
22   FileObj fileObj;
23   MainWindow mainWindow;
24   public UnfoldingSimulator() {
25   }
26
27   void run() {
28     int i;
29     String fileName;
30
31     // init
32     paper =  new UnfoldingPaper();
33     fileObj = new FileObj();
34     mainWindow = new UnfoldingWindow();
35
36     // load
37     fileName = String.format(FILENAME, 1);
38     fileObj.load(fileName, paper);
39
40     ((UnfoldingWindow)mainWindow).run(paper);
41
42   }
43
44
45   public static void main(String[] args) {
46     // TODO Auto-generated method stub
47     UnfoldingSimulator simulator = new UnfoldingSimulator();
48     simulator.run();
49   }
50 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.boxcorner;
2
3  import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
4  import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
       .MainWindowForFoldingRobotOrigami;
5
6  public class BoxCornerSimulator {
7    final static double DEFULT_STEP = 0.02f;
8    final static String FILENAME = "C:\\foldingdata\\boxCorner\\BoxCornerData
       .csv";
9
10   public void run(){
11     int i;
12
13     double theta;
14     double cos;
15     double sin;
16
17     double a;
18     double b;
19     double c;
20
21
22     int numberOfLevels;
23
24     numberOfLevels = ((int)((Math.PI) /2  / DEFULT_STEP)) + 1;
25
26     double [] origin = new double[3];
27
28     System.out.format("c'-d'-e\n");
29     System.out.format("|'\\| /|\n");
30     System.out.format("b'-o'-d\n");
31     System.out.format("| /|'\\|\n");
32     System.out.format("a -b -c\n");
33     System.out.format("\n");
34
35     System.out.format("7 -6 -5\n");
36     System.out.format("|'\\| /|\n");
37     System.out.format("8 -0 -4\n");
38     System.out.format("| /| \\|\n");
39     System.out.format("1 -2 -3\n");
40     System.out.format("\n");
41     System.out.format("\n");
42
43
44     double [][][] points = new double [numberOfLevels][9][3];
45     double [][] angles = new double [numberOfLevels][9];
46
47     double [][][] pointOfMagnets = new double [numberOfLevels][2][3];
48     Vector [][] unitVectorOfMagnets = new Vector [numberOfLevels][2];
49
50     Vector v1, v2;
51
52     double [] distancesBetweenMagnets = new double [numberOfLevels];
53
54     theta=Math.PI + DEFULT_STEP;
55     for (i = 0; i < numberOfLevels; i++){
56
57       theta-=DEFULT_STEP;
58       cos=Math.cos(Math.PI - theta);
59       sin=Math.sin(Math.PI - theta);
60
61       // 0 = {        0.0f,          0.0f,          0.0f}
62       // 1 = {       -1.0f,         -1.0f,          0.0f}
63       // 2 = {        0.0f,         -1.0f,          0.0f}
64       // 3 = {COS(PI-TH2),         -1.0f,  SIN(PI-TH2)}
65       // 4 = {COS(PI-TH2),          0.0f,  SIN(PI-TH2)}
66       // 5 = { (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta]^3 -
67 //           (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta]^3 -
68 //            Cos[theta] Sin[theta]^2 + \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2]
       Sin[theta]^2 +
69 //              4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 +
70 //              4 Sin[theta]^4 + 4 Sqrt[2] Sin[theta]^4 -
71 //              4 Cos[theta]^2 Sin[theta]^4 - 2 Sin[theta]^6))/(2 (Cos[
72 //              theta]^2 + 2 Sin[theta]^2)),
73 //           (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta]^3 -
74 //              Cos[theta] Sin[
75 //              theta]^2 + \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2] Sin[theta]^2
       +
76 //              4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 +
77 //              4 Sin[theta]^4 + 4 Sqrt[2] Sin[theta]^4 -
78 //              4 Cos[theta]^2 Sin[theta]^4 - 2 Sin[theta]^6))/(2 (Cos[
       theta]^2 +
79 //              2 Sin[theta]^2)),
```

```
80 //          \[Sqrt](Sqrt[2] - (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta
       ]^3[theta]
81 //              Cos[theta] Sin[1]^2 + \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2] Sin
       [theta]^2 +
82 //              4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 + 4
        Sin[theta]^4 +
83 //              4 Sqrt[2] Sin[theta]^4 - 4 Cos[theta]^2 Sin[theta]^4 -
84 //              2 Sin[theta]^6))^2/(2 (Cos[theta]^2 + 2 Sin[theta]^2)^2))
85
86        // 6 = {        0.0,    COS(PI-Th2),    SIN(PI-TH2)}
87        // 7 = {       -1.0,    COS(PI-Th2),    SIN(PI-TH2)}
88        // 8 = {        0.0,          -1.0,          0.0}
89
90        // 0 = {        0.0,           0.0,          0.0}
91        points[i][0][0] =  0.0;
92        points[i][0][1] =  0.0;
93        points[i][0][2] =  0.0;
94
95        // 1 = {       -1.0,          -1.0,          0.0}
96        points[i][1][0] = -1.0;
97        points[i][1][1] = -1.0;
98        points[i][1][2] =  0.0;
99
100       // 2 = {        0.0,          -1.0,          0.0}
101       points[i][2][0] =  0.0;
102       points[i][2][1] = -1.0;
103       points[i][2][2] =  0.0;
104
105       // 3 = {COS(PI - theta),        -1.0, SIN(PI-theta)}
106       //      == -1 * COS(theta)           SIN(Theta)
107       points[i][3][0] = cos;
108       points[i][3][1] = -1.0;
109       points[i][3][2] = sin;
110
111       // 4 = {COS(PI-Th2),          0.0, SIN(PI-TH2)}
112       //      ==  -1 * COS(theta)
113       points[i][4][0] =  cos;
114       points[i][4][1] =  0.0;
115       points[i][4][2] =  sin;
116
117       a = points[i][4][0];
118       b = points[i][4][1];
119       c = points[i][4][2];
120
121
122       // 5
123 //      {{z -> (4 c - 4 a^2 c - 4 b^2 c -
124 //              4 c^3 - \[Sqrt]((-4 c + 4 a^2 c + 4 b^2 c + 4 c^3)^2 -
125 //                 4 (-a^2 - 2 a b - b^2 - 2 c^2) (-2 + (4 + Sqrt[2]) a^2 -
126 //                    2 a^4 + 2 Sqrt[2] a b + (4 + Sqrt[2]) b^2 - 4 a^2 b^2
       -
127 //                    2 b^4 + 4 c^2 - 4 a^2 c^2 - 4 b^2 c^2 -
128 //                    2 c^4)))/(2 (-a^2 - 2 a b - b^2 - 2 c^2))},
129
130
131 //       {z -> (4 c -
132 //              4 a^2 c - 4 b^2 c -
133 //              4 c^3 + \[Sqrt]((-4 c + 4 a^2 c + 4 b^2 c + 4 c^3)^2 -
134 //                 4 (-a^2 - 2 a b - b^2 - 2 c^2) (-2 + (4 + Sqrt[2]) a^2 -
135 //                    2 a^4 + 2 Sqrt[2] a b + (4 + Sqrt[2]) b^2 - 4 a^2 b^2
       -
136 //                    2 b^4 + 4 c^2 - 4 a^2 c^2 - 4 b^2 c^2 -
137 //                    2 c^4)))/(2 (-a^2 - 2 a b - b^2 - 2 c^2))}}
138
139 /** /
140       points[i][5][2] = (4 * c -
141               4 * Math.pow(a,2) * c - 4 * Math.pow(b,2) * c -
142               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) -
143                 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) -
144                    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) -
145                    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) -
146                    2 * Math.pow(c,4))))/(2 * (-1 * Math.pow(a,2) - 2 * a *
                            b - Math.pow(b,2) - 2 * Math.pow(c,2) ));
147
148 /** /
149         // 5 = { (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta]^3 - Cos[theta]
                Sin[theta]^2 +
150 //              \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2] Sin[theta]^2 +
151 //                4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 +
152 //                4 Sin[theta]^4 + 4 Sqrt[2] Sin[theta]^4 -
153 //                4 Cos[theta]^2 Sin[theta]^4 - 2 Sin[theta]^6) )) /
154 //              (2 (Cos[theta]^2 +
155 //                2 Sin[theta]^2 ) ),
156         points[i][5][0] =  (cos - sqrt2 * cos - Math.pow(cos,3) - cos * Math.
                pow(sin, 2) +
157           Math.sqrt( -6 * Math.pow(sin, 2) + 4 * sqrt2 * Math.pow(sin, 2) +
158           4 * Math.pow(cos, 2) * Math.pow(sin , 2) - 2 * Math.pow(cos, 4) *
                  Math.pow(sin, 2) +
159           4 * Math.pow(sin, 4) + 4 * sqrt2 * Math.pow(sin, 4) -
160           4 * Math.pow(cos, 2) * Math.pow(sin, 4) - 2 * Math.pow(sin, 6) ))/
161           (2 *  (Math.pow(cos, 2) +
162           2 * Math.pow(sin, 2) ));
163
164 //      (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta]^3 - Cos[theta] Sin[
              theta]^2 +
165 //        \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2] Sin[theta]^2 +
166 //          4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 +
167 //          4 Sin[theta]^4 + 4 Sqrt[2] Sin[theta]^4 -
168 //          4 Cos[theta]^2 Sin[theta]^4 - 2 Sin[theta]^6) )) /
169 //        (2 (Cos[theta]^2 +
170 //          2 Sin[theta]^2 ) ),
171         points[i][5][1] =  (cos - sqrt2 * cos - Math.pow(cos,3) - cos * Math.
                pow(sin, 2) +
172           Math.sqrt( -6 * Math.pow(sin, 2) + 4 * sqrt2 * Math.pow(sin, 2) +
173           4 * Math.pow(cos, 2) * Math.pow(sin , 2) - 2 * Math.pow(cos, 4) *
                  Math.pow(sin, 2) +
174           4 * Math.pow(sin, 4) + 4 * sqrt2 * Math.pow(sin, 4) -
175           4 * Math.pow(cos, 2) * Math.pow(sin, 4) - 2 * Math.pow(sin, 6) ))/
176           (2 *  (Math.pow(cos, 2) +
177           2 * Math.pow(sin, 2) ));
178
179
180 //          \[Sqrt](Sqrt[2] - (Cos[theta] - Sqrt[2] Cos[theta] - Cos[theta
       ]^3 - Cos[theta] Sin[theta]^2 +
181 //              \[Sqrt](-6 Sin[theta]^2 + 4 Sqrt[2] Sin[theta]^2 +
182 //              4 Cos[theta]^2 Sin[theta]^2 - 2 Cos[theta]^4 Sin[theta]^2 + 4
        Sin[theta]^4 +
183 //              4 Sqrt[2] Sin[theta]^4 - 4 Cos[theta]^2 Sin[theta]^4 -
184 //              2 Sin[theta]^6))^2/(2 (Cos[theta]^2 + 2 Sin[theta]^2)^2))
185       points[i][5][2] =
186         Math.sqrt(sqrt2 - Math.pow(cos - sqrt2 * cos - Math.pow(cos, 3) -
                cos * Math.pow(sin, 2) +
187         Math.sqrt(-6 * Math.pow(sin, 2) + 4 * sqrt2 * Math.pow(sin, 2) +
188         4 * Math.pow(cos,2) * Math.pow(sin,2) - 2 * Math.pow(cos, 4) *
                Math.pow(sin, 2) + 4 * Math.pow(sin, 4) +
189         4 * sqrt2 * Math.pow(sin, 4) - 4 * Math.pow(cos, 2) * Math.pow(sin,
                4) -
190         2 * Math.pow(sin, 6)), 2) / (2 * Math.pow((Math.pow(cos, 2) + 2 *
                Math.pow(sin, 2)),2) ));
191 /**/
```

```
192
193         // 5 = {            , (-1 + a^2 + b^2 + c^2 - c* z) / (a + b)
194 //                           , (-1 + a^2 + b^2 + c^2 - c* z) / (a + b)
195 //                           , (4 c - 4 a^2 c - 4 b^2 c -
196 //                          4 c^3 + \/Sqrt[((-4 c + 4 a^2 c + 4 b^2 c + 4 c^3)
        ^2 -
197 //                                    4 (-a^2 - 2 a b - b^2 - 2 c^2) (-2 + (4 +
        Sqrt[2]) a^2 -
198 //                                       2 a^4 + 2 Sqrt[2] a b + (4 + Sqrt[2]) b
        ^2 - 4 a^2 b^2 -
199 //                                       2 b^4 + 4 c^2 - 4 a^2 c^2 - 4 b^2 c^2 -
        2 c^4)))/(2 (-a^2 -
200 //                                   2 a b - b^2 - 2 c^2))
201 //                                  )           ,
202           a = points[i][4][0];
203           b = points[i][4][1];
204           c = points[i][4][2];
205
206 //          x=(a + a^3 + b + a^2 b + a b^2 + b^3 + a c^2 + b c^2 -
207 //                Sqrt[2] Sqrt[-c^2 (1 + a^4 - 8 a b + b^4 - 6 c^2 + c^4 +
208 //                     2 b^2 (-1 + c^2) + 2 a^2 (-1 + b^2 + c^2))])/(2 (a
        ^2 + 2 a b +
209 //                     b^2 + 2 c^2))
210           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) -
211               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) +
212               2 * Math.pow(b,2) * (-1 + Math.pow(c,2)) + 2 * Math.pow(a,2)
                      * (-1 + Math.pow(b,2) + Math.pow(c,2)))))/
213               (2 * (Math.pow(a,2) + 2 * a * b + Math.pow(b,2) + 2 * Math.
                      pow(c,2)));
214           points[i][5][1] = points[i][5][0];
215           points[i][5][2] =  Math.sqrt(2) * Math.sqrt(1 - Math.pow(points[i
                ][5][0],2));
216
217 /** /
218           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) +
219                 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) +
220                   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) -
221                   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)));
222           points[i][5][1] = points[i][5][0];
223           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);
224 /** /
225           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) +
226                 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) +
227                   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) -
228                   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)));
```

```
229           points[i][5][1] =  points[i][5][0];
230           points[i][5][2] = Math.sqrt(2-2*Math.sqrt(points[i][5][0]));
231 /** /
232         points[i][5][2] =  (4 * c - 4 * Math.pow(a,2) * c - 4 * Math.pow(b,2)
                * c - 4 * Math.pow(c,3) -
233                   Math.sqrt(Math.pow((-4 * c + 4 * Math.pow(a,2) * c + 4 *
                        Math.pow( b,2) *c + 4 * Math.pow(c,3)),2) -
234                     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) -
235                       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) -
236                       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))))/
237                       (2 * (-1 * Math.pow(a,2) - 2 * a * b - Math.pow(b
                            ,2) - 2  * Math.pow(c,2)));
238
239         points[i][5][0] = (-1 + Math.pow(a,2) + Math.pow(b,2) + Math.pow(c,2)
                - c* points[i][5][2]) / (a + b);
240         points[i][5][1] = points[i][5][0];
241 /**/
242         // 6 = {        0.0, COS(PI-theta), SIN(PI-theta)}
243         //             == -1 * COS(theta) SIN(Theta)
244         points[i][6][0] =  0.0;
245         points[i][6][1] =  cos;
246         points[i][6][2] =  sin;
247
248         // 7 = {       -1.0, COS(PI-theta), SIN(PI-theta)}
249         //             == -1 * COS(theta) SIN(Theta)
250         points[i][7][0] = -1.0;
251         points[i][7][1] =  cos;
252         points[i][7][2] =  sin;
253
254         // 8 = {       -1.0,          0.0,          0.0}
255         points[i][8][0] = -1.0;
256         points[i][8][1] =  0.0;
257         points[i][8][2] =  0.0;
258
259       }
260
261       // get angles
262       for (i = 0; i < numberOfLevels; i++){
263         angles[i][0] = Math.PI;
264
265         origin[0] = points[i][1][0] / 2;
266         origin[1] = points[i][1][1] / 2;
267         origin[2] = points[i][1][2] / 2;
268         angles[i][1] = getAngle(origin , points[i][8], points[i][2]);
269
270         angles[i][2] = getAngle(points[i][2],points[i][1], points[i][3]);
271
272         origin[0] = points[i][3][0] / 2;
273         origin[1] = points[i][3][1] / 2;
274         origin[2] = points[i][3][2] / 2;
275         angles[i][3] = getAngle(origin,points[i][2], points[i][4]);
276
277         angles[i][4] = getAngle(points[i][4],points[i][3], points[i][5]);
278
279         origin[0] = points[i][5][0] / 2;
280         origin[1] = points[i][5][1] / 2;
281         origin[2] = points[i][5][2] / 2;
282         angles[i][5] = getAngle(origin , points[i][4], points[i][6]);
283
284         angles[i][6] = getAngle(points[i][6],points[i][5], points[i][7]);
285
286         origin[0] = points[i][7][0] / 2;
287         origin[1] = points[i][7][1] / 2;
```

```
288         origin[2] = points[i][7][2] / 2;
289         angles[i][7] = getAngle(origin, points[i][6], points[i][8]);
290
291         angles[i][8] = getAngle(points[i][8],points[i][7], points[i][1]);
292
293
294       }
295
296
297       // get point of the magnet
298       // get unitVector of the magnet
299       // get distance from magnets
300
301       for (i = 0; i < numberOfLevels; i++){
302         pointOfMagnets[i][0][0] = points[i][4][0] / 2;
303         pointOfMagnets[i][0][1] = points[i][4][1] / 2;
304         pointOfMagnets[i][0][2] = points[i][4][2] / 2;
305
306         pointOfMagnets[i][1][0] = points[i][6][0] / 2;
307         pointOfMagnets[i][1][1] = points[i][6][1] / 2;
308         pointOfMagnets[i][1][2] = points[i][6][2] / 2;
309
310
311
312         unitVectorOfMagnets[i][0]= new Vector();
313         unitVectorOfMagnets[i][1]= new Vector();
314
315         v1 = new Vector();
316         v2 = new Vector();
317
318
319         v1.setXYZ((float)points[i][3][0],
320             (float)points[i][3][1],
321             (float)points[i][3][2] );
322         v1.invert();
323
324         unitVectorOfMagnets[i][0].setXYZ((float)points[i][4][0],
325             (float)points[i][4][1],
326             (float)points[i][4][2] );
327
328         unitVectorOfMagnets[i][0].addVector(v1);
329         unitVectorOfMagnets[i][0] = unitVectorOfMagnets[i][0].getUnitVector()
                ;
330
331         v2.setXYZ((float)points[i][7][0],
332             (float)points[i][7][1],
333             (float)points[i][7][2] );
334         v2.invert();
335
336         unitVectorOfMagnets[i][1].setXYZ((float)points[i][6][0],
337             (float)points[i][6][1],
338             (float)points[i][6][2] );
339         unitVectorOfMagnets[i][1].addVector(v2);
340         unitVectorOfMagnets[i][1] = unitVectorOfMagnets[i][1].getUnitVector()
                ;
341
342         distancesBetweenMagnets[i] = getDistance(points[i][4], points[i][6])
                / 2;
343
344       }
345
346
347       FileBoxCornerAngleData fileBoxCornerAngleData = new
              FileBoxCornerAngleData();
348       if (fileBoxCornerAngleData.build(FILENAME, points, angles,
            pointOfMagnets, unitVectorOfMagnets, distancesBetweenMagnets)){
349
350         System.out.format("%s is created\n", FILENAME);
351
352       } else {
353         System.out.format("error during %s is being creating\n", FILENAME);
354
355       }
356
357       MonitorBoxCornerSimulator monitor = new MonitorBoxCornerSimulator ();
358       monitor.run(points);
359
360     }
361
362     private double getAngle(double [] origin, double [] v1, double[] v2) {
363       int i;
364       double angle = 0;
365       double dot = 0;
366
367       double [] uv1 = new double [3];
368       double [] uv2 = new double [3];
369
370       double uv1Length = 0;
371       double uv2Length = 0;
372
373
374       for(i=0; i<3 ; i++) {
375         uv1[i] = v1[i] - origin[i] ;
376         uv2[i] = v2[i] - origin[i] ;
377
378         uv1Length += Math.pow(uv1[i], 2);
379         uv2Length += Math.pow(uv2[i], 2);
380       }
381
382
383       uv1Length = Math.sqrt(uv1Length);
384       uv2Length = Math.sqrt(uv2Length);
385
386       dot = 0;
387       for(i=0; i<3 ; i++) {
388         uv1[i] = uv1[i] / uv1Length;
389         uv2[i] = uv2[i] / uv2Length;
390
391         dot += uv1[i] * uv2[i];
392       }
393
394       if (dot>1) {
395         dot = 1;
396       }
397       if (dot<-1) {
398         dot = -1;
399       }
400
401       angle = Math.acos(dot);
402       return angle;
403     }
404
405     public double getDistance(double v1[], double v2[]) {
406         return Math.sqrt(Math.pow(v2[0]- v1[0], 2) + Math.pow(v2[1]- v1[1],
                2) + Math.pow(v2[2]- v1[2] ,2));
407     }
408
409
410     public static void main(String[] args) {
411
412
413
414       BoxCornerSimulator boxCornerSimulator = new BoxCornerSimulator();
415       boxCornerSimulator.run();
416
417     }
418 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.boxcorner;
2
3  import java.io.BufferedWriter;
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.io.FileWriter;
7  import java.io.IOException;
8
9  import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
10
11 public class FileBoxCornerAngleData {
12   public boolean build(String fileName, double [][][] points, double [][]
         angles, double[][][] pointOfMagnets, Vector [][] unitVectorOfMagnets
         ,double [] distantBetweenMagnets ) {
13     int i;
14     int j;
15     int k;
16
17
18     int numberOfLevels;
19     int numberOfEdges = 8;
20
21
22     File file = new File(fileName);
23
24     numberOfLevels = points.length;
25
26     try {
27       boolean success = file.createNewFile();
28       if (success) {
29         // File did not exist and was created
30       } else {
31
32       }
33     } catch (IOException e) {
34       // TODO Auto-generated catch block
35       e.printStackTrace();
36     }
37
38     String bufferLine  = new String();
39
40     try {
41       BufferedWriter bufferedWriter = new BufferedWriter (new FileWriter (
             file));
42
43       // print points
44       bufferLine = String.format("# %d levels", numberOfLevels);
45       // print to file
46       bufferedWriter.write(bufferLine);
47       bufferedWriter.newLine();
48
49       bufferLine = String.format("# %d edges", 8);
50       // print to file
51       bufferedWriter.write(bufferLine);
52       bufferedWriter.newLine();
53
54       for (i = 0; i < numberOfLevels; i++) {
55
56         for (j=0 ; j < 9 ; j++) {
57           bufferLine = String.format("p");
58           bufferLine += String.format(", %d, %f, %f, %f", j, points[i][j
                 ][0]
59                                                          , points[i][j
                                                              ][1]
60                                                          , points[i][j
                                                              ][2]);
61           // print to file
62           bufferedWriter.write(bufferLine);
63           bufferedWriter.newLine();
64         }
65       }
66
67       // print angles
68       bufferLine = String.format("# %d levels", numberOfLevels);
69
70       // print to file
71       bufferedWriter.write(bufferLine);
72       bufferedWriter.newLine();
73
74       bufferLine = String.format("# %d edges", 8);
75       // print to file
76       bufferedWriter.write(bufferLine);
77       bufferedWriter.newLine();
78
79       // angle[level][edgeNumber]
80       for (i = 0; i < numberOfLevels; i++) {
81         for (j = 1; j < numberOfEdges + 1; j++) {
82           bufferLine = String.format("a");
83           bufferLine += String.format(", %d, %f", j, angles[i][j]);
84
85           // print to file
86           bufferedWriter.write(bufferLine);
87           bufferedWriter.newLine();
88         }
89       }
90
91       bufferLine = String.format("# %d edges", 8);
92       // print to file
93       bufferedWriter.write(bufferLine);
94       bufferedWriter.newLine();
95
96       // angle[level][edgeNumber]
97       for (i = 0; i < numberOfLevels; i++) {
98         bufferLine = String.format("#g");
99         bufferLine += String.format(", %d", i);
100        for (j = 1; j < numberOfEdges + 1; j++) {
101
102          bufferLine += String.format(", %f", angles[i][j]);
103
104
105        }
106        // print to file
107        bufferedWriter.write(bufferLine);
108        bufferedWriter.newLine();      }
109
110
111      bufferLine = String.format("# pointOfMagnet" );
112      // print to file
113      bufferedWriter.write(bufferLine);
114      bufferedWriter.newLine();
115
116      bufferLine = String.format("# Magnet1'sX,Y,Z,Magnet2'sX,Y,z" );
117      // print to file
118      bufferedWriter.write(bufferLine);
119      bufferedWriter.newLine();
120
121      // point of magnet
122      for (i = 0; i < numberOfLevels; i++) {
123        bufferLine = String.format("#pm");
124        bufferLine += String.format(", %d", i);
125        for (j=0; j<2; j++) {
126          for (k=0;k < 3;k++){
127            bufferLine += String.format(", %f", pointOfMagnets[i][j][k]);
128          }
129        }
130        // print to file
131        bufferedWriter.write(bufferLine);
132        bufferedWriter.newLine();
```

```
133         }
134         bufferLine = String.format("# UnitVectorOfMagnet1'sX,Y,Z,
                UnitVectorOfMagnet2'sX,Y,z" );
135         // print to file
136         bufferedWriter.write(bufferLine);
137         bufferedWriter.newLine();
138
139         // unitVector of magnet
140         for (i = 0; i < numberOfLevels; i++) {
141           bufferLine = String.format("#uv");
142           bufferLine += String.format(", %d", i);
143           for (j=0; j<2; j++) {
144             bufferLine += String.format(", %f", unitVectorOfMagnets[i][j].
                    getX());
145             bufferLine += String.format(", %f", unitVectorOfMagnets[i][j].
                    getY());
146             bufferLine += String.format(", %f", unitVectorOfMagnets[i][j].
                    getZ());
147           }
148           // print to file
149           bufferedWriter.write(bufferLine);
150           bufferedWriter.newLine();
151         }
152
153
154         bufferLine = String.format("# %d distant", 1);
155         // print to file
156         bufferedWriter.write(bufferLine);
157         bufferedWriter.newLine();
158
159         // distants level)
160         for (i = 0; i < numberOfLevels; i++) {
161           bufferLine = String.format("#d");
162           bufferLine += String.format(", %d", i);
163           bufferLine += String.format(", %f", distantBetweenMagnets[i]);
164
165
166           // print to file
167           bufferedWriter.write(bufferLine);
168           bufferedWriter.newLine();
169         }
170
171         // file close
172         bufferedWriter.close();
173       } catch (FileNotFoundException e) {
174         e.printStackTrace();
175         return false;
176       } catch (IOException e) {
177         e.printStackTrace();
178         return false;
179       }
180       return true;
181     }
182 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.boxcorner;
2
3
4  import java.awt.Dimension;
5  import java.awt.Frame;
6  import java.awt.event.*;
7
8  import javax.media.opengl.*;
9
10 import sun.text.normalizer.UProperty;
11
12 import com.drancom.programmableMatter.folding.controller.paper.Paper;
13 import com.drancom.programmableMatter.folding.controller.paper.Point;
14 import com.drancom.programmableMatter.folding.controller.paper.Polygon;
15 import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
16 import com.sun.opengl.util.*;
17
18 public class MonitorBoxCornerSimulator implements GLEventListener,
       MouseListener, MouseMotionListener {
19   public final static double ZOOM_MAGNIFICATION = 4;
20   public final static float LINEWIDTH = 2;
21
22   double [][][][] points;
23
24   // light
25   // float pos0[] = { -100.0f, 130.0f, 150.0f, 1.0f };
26   public final static float pos0[] = { -100.0f, 130.0f, 150.0f, 1.0f };
27   public final static float ambientLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
28   public final static float diffuseLight[] = { 0.25f, 0.25f, 0.25f, 1.0f};
29   public final static float specular[] = { 0.25f, 0.25f, 0.25f, 1.0f};
30
31   public final static float WHITE[]  = { 1.0f, 1.0f, 1.0f, 0.2f };
32   public final static float RED[]    = { 1.0f, 0.0f, 0.0f, 1.0f };
33   public final static float GREEN[]  = { 0.0f, 1.0f, 0.0f, 1.0f };
34   public final static float YELLOW[] = { 1.0f, 1.0f, 0.0f, 1.0f };
35   public final static float BLUE[]   = { 0.0f, 0.0f, 1.0f, 0.0f };
36   public final static float BLACK[]  = { 0.0f, 0.0f, 0.0f, 1.0f };
37
38   final static int SPEED_OF_ANIMATION = 10; //10 is default
39   int counterForSpeedOfAnimation = 0;
40
41   boolean isAnimating=false;
42
43   public void run(double[][][][] points) {
44     this.points = points;
45     paperGlId = new int[points.length];
46
47
48     Frame frame = new Frame("Play Window - Programmable Matter by Folding")
           ;
49       GLCanvas canvas = new GLCanvas();
50
51     canvas.addGLEventListener(this);
52     frame.add(canvas);
53     frame.setSize(800, 800);
54     final Animator animator = new Animator(canvas);
55     frame.addWindowListener(new WindowAdapter() {
56       public void windowClosing(WindowEvent e) {
57           // Run this on another thread than the AWT event queue to
58         // make sure the call to Animator.stop() completes before
59         // exiting
60         new Thread(new Runnable() {
61           public void run() {
62             animator.stop();
63             System.exit(0);
64           }
65         }).start();
66           }
67     });
68       frame.show();
69       animator.start();
70   }
71
72   private float view_rotx =  -20.0f, view_roty =  -10.0f, view_rotz = 10.0f
         ;
73
74   private int paperGlId[];
75   private int currentIndex_PaperGlId;
76
77   private boolean isFolding = true;
78
79   public boolean isFolding() {
80     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;
          }
        }
      }
    }
```

```
217   }
218
219   public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
          boolean deviceChanged) {}
220
221   public void buildGlPaper(GL gl, double [][] points){
222     int i;
223     int j;
224     int numberOfPoint = points.length;
225
226
227     gl.glShadeModel(GL.GL_FLAT);
228
229     gl.glNormal3f(0.0f, 0.0f, 1.0f);
230
231     /* draw polygon */
232     for (i=0; i<points.length - 1; i++){
233
234       gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WHITE, 0);
235
236       gl.glBegin(GL.GL_TRIANGLES);
237       gl.glVertex3d(points[0][0] * ZOOM_MAGNIFICATION
238             , points[0][1] * ZOOM_MAGNIFICATION
239             , points[0][2] * ZOOM_MAGNIFICATION);
240
241       gl.glVertex3d(points[1 + i%8][0] * ZOOM_MAGNIFICATION
242             , points[1 + i%8][1] * ZOOM_MAGNIFICATION
243             , points[1 + i%8][2] * ZOOM_MAGNIFICATION);
244
245       gl.glVertex3d(points[1 + (i + 1)% 8][0] * ZOOM_MAGNIFICATION
246             , points[1 + (i + 1)% 8][1] * ZOOM_MAGNIFICATION
247             , points[1 + (i + 1)% 8][2] * ZOOM_MAGNIFICATION);
248 /**/
249       gl.glVertex3d(points[0][0] * ZOOM_MAGNIFICATION
250           , points[0][1] * ZOOM_MAGNIFICATION
251           , points[0][2] * ZOOM_MAGNIFICATION);
252
253       gl.glVertex3d(points[1 + (i + 1)% 8][0] * ZOOM_MAGNIFICATION
254             , points[1 + (i + 1)% 8][1] * ZOOM_MAGNIFICATION
255             , points[1 + (i + 1)% 8][2] * ZOOM_MAGNIFICATION);
256
257       gl.glVertex3d(points[1 + i%8][0] * ZOOM_MAGNIFICATION
258           , points[1 + i%8][1] * ZOOM_MAGNIFICATION
259           , points[1 + i%8][2] * ZOOM_MAGNIFICATION);
260 /**/
261       gl.glEnd();
262
263     }
264
265     /* draw lines */
266
267     for (i=0; i<points.length-1; i++) {
268
269         gl.glLineWidth(LINEWIDTH);
270
271
272
273       gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, WHITE, 0);
274
275       gl.glBegin(GL.GL_LINES);
276
277       gl.glVertex3d((float)points[0][0]*ZOOM_MAGNIFICATION,
278           (float)points[0][1]*ZOOM_MAGNIFICATION,
279           (float)points[0][2]*ZOOM_MAGNIFICATION);
280
281       gl.glVertex3d(points[1 + i%8][0] * ZOOM_MAGNIFICATION
282           , points[1 + i%8][1] * ZOOM_MAGNIFICATION
283           , points[1 + i%8][2] * ZOOM_MAGNIFICATION);
284
285       gl.glVertex3d(points[1 + i%8][0] * ZOOM_MAGNIFICATION
286           , points[1 + i%8][1] * ZOOM_MAGNIFICATION
287           , points[1 + i%8][2] * ZOOM_MAGNIFICATION);
288
289       gl.glVertex3d(points[1 + (i + 1)% 8][0] * ZOOM_MAGNIFICATION
290           , points[1 + (i + 1)% 8][1] * ZOOM_MAGNIFICATION
291           , points[1 + (i + 1)% 8][2] * ZOOM_MAGNIFICATION);
292
293       gl.glEnd();
294
295     }
296   }
297
298     // Methods required for the implementation of MouseListener
299   public void mouseEntered(MouseEvent e) {}
300   public void mouseExited(MouseEvent e) {}
301   public void mousePressed(MouseEvent e) {
302
303     prevMouseX = e.getX();
304     prevMouseY = e.getY();
305     if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
306       mouseRButtonDown = true;
307     }
308
309     if (mouseRButtonDown == true) {
310       isAnimating = true;
311     }
312   }
313
314   public void mouseReleased(MouseEvent e) {
315     if ((e.getModifiers() & e.BUTTON3_MASK) != 0) {
316       mouseRButtonDown = false;
317       }
318   }
319
320   public void mouseClicked(MouseEvent e) {}
321
322   // Methods required for the implementation of MouseMotionListener
323   public void mouseDragged(MouseEvent e) {
324     if (mouseRButtonDown == false) {
325       int x = e.getX();
326       int y = e.getY();
327       Dimension size = e.getComponent().getSize();
328
329       float thetaY = 360.0f * ( (float)(x-prevMouseX)/(float)size.width);
330       float thetaX = 360.0f * ( (float)(prevMouseY-y)/(float)size.height);
331
332       prevMouseX = x;
333       prevMouseY = y;
334
335       view_rotx += thetaX;
336       view_roty += thetaY;
337     }else {
338
339     }
340   }
341
342   public void mouseMoved(MouseEvent e) {
343
344   }
345 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamiWithStepFunction;
2
3  import com.drancom.programmableMatter.folding.controller.paper.Paper;
4  import com.drancom.programmableMatter.folding.dataFile.FilePlan;
5  import com.drancom.programmableMatter.folding.origami.planner.Plan;
6  import com.drancom.programmableMatter.folding.origami.planner.Planner;
```

```
7  import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
       .FilePlanForOrigami;
8  import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
       .PlanForOrigami;
9  import com.drancom.programmableMatter.stepFunction.StepFunction;
10
11 public class PlanerForOrigamiWithStepFunction implements Planner {
12 //   Default: percentage of level0 = 33, percentage of level1 = 66,
       percentage of level2 = 100
13 //   NoiseNumber = 0.01
14   final static float EPS = 0.5f;
15
16   final static float PERCENTAGE_OF_LEVEL0 = 0.33f;
17   final static float PERCENTAGE_OF_LEVEL1 = 0.66f;
18   final static float PERCENTAGE_OF_LEVEL2 = 1.0f;
19
20   final static float NOISE_NUMBER= 0.01f;
21
22   PlanForOrigami planForOrigami;
23
24   @Override
25   public void build(Paper[] papers) {
26
27 //     Input:  Angledata[numberOfEdge][time]
28     float angleData[][] = new float[papers[0].getNumberOfEdges()][papers.
           length];
29
30 //     Output: Plan[phase][n], numberOfPhase
31     this.planForOrigami = new PlanForOrigami();
32     int plan[][] = new int[papers.length][papers[0].getNumberOfEdges()];
33     int numberOfPhases;
34
35     float angleLevel[][] = new float [papers[0].getNumberOfEdges()][papers.
           length];
36     int angleDifference[][] = new int [papers[0].getNumberOfEdges()][papers
           .length];
37
38     int phase;
39     boolean isBuildingPlan;
40     boolean isAllZero;
41
42     int i;
43     int j;
44     int k=0;
45
46     for (i=0; i < papers[0].getNumberOfEdges(); i++){
47       for (j=0; j < papers.length; j++) {
48         angleData[i][j] = papers[j].getLine(i).getAngle();
49         angleLevel[i][j] = 0;
50         if (angleData[i][j] <0) {
51           k++;
52         }
53       }
54     }
55
56
57 //     1-6 StepFunction
58 //     1. for i = 0 to NumberOfEdge
59 //     2.     angleLevel[i] <- stepFunction(angledata[i], EPS) // EPS = 0.5
60     for(i=0; i < papers[0].getNumberOfEdges(); i++) {
61       angleLevel[i] =  StepFunction.stepFunction(angleData[i], EPS);
62     }
63
64 //     3. for i = 0 to numberOfEdge
65 //     4.       for j=0 to papers.length -1
66 //     5.         angleDifference [i][j] =0;
67 //     6.         if (angleLevel[i][j] > 0 && angleLevel[i][j] != angleLevel[i
       ][0] && angleLevel[i][j] != angleLevel[i][angleLevel[i].length - 1 ])
68 //     7.           angleDifference[i][j] = 1;
69 //     8.         if (angleLevel[i][j] > 0 && angleLevel[i][j] - angleLevel[i][
       j+1] != 0 ) {
70 //     9.           angleDifference[i][j] = 1;
71 //     10.        if (angleLevel[i][j] < 0 && angleLevel[i][j] != angleLevel[i
       ][0] && angleLevel[i][j] != angleLevel[i][angleLevel[i].length - 1 ])
72 //     11.          angleDifference[i][j] = -1;
73 //     12.        if (angleLevel[i][j] < 0 && angleLevel[i][j] - angleLevel[i][
       j+1] != 0 ) {
74 //     13.          angleDifference[i][j] = -1;
75
76     for (i=0; i < papers[0].getNumberOfEdges(); i++){
77       for (j=0; j < papers.length-1; j++) {
78
79         angleDifference[i][j] = 0;
80         if (angleLevel[i][j] > 0 && angleLevel[i][j] != angleLevel[i][0] &&
               angleLevel[i][j] != angleLevel[i][angleLevel[i].length - 1 ])
               {
81           angleDifference[i][j] = 1;
82         }
83         if (angleLevel[i][j] > 0 && angleLevel[i][j] - angleLevel[i][j+1]
               != 0 ) {
84           angleDifference[i][j] = 1;
85         }
86         if (angleLevel[i][j] < 0 && angleLevel[i][j] != angleLevel[i][0] &&
               angleLevel[i][j] != angleLevel[i][angleLevel[i].length - 1 ])
               {
87           angleDifference[i][j] = -1;
88         }
89         if (angleLevel[i][j] < 0 && angleLevel[i][j] - angleLevel[i][j+1]
               != 0 ) {
90           angleDifference[i][j] = -1;
91         }
92       }
93     }
94
95 //     14. Phase <- 0
96 //     15. numberOfPhase
97 //     16. isBuildingPlan <- false
98 //     17. IsAllZero <- true
99     phase = 0;
100    numberOfPhases = 1;
101    isBuildingPlan = false;
102    isAllZero = true;
103
104 //     18. Plan[1..][1..numberOfEdge] <- 0
105    for (i=0; i < papers.length; i++) {
106      for (j=0; j < papers[0].getNumberOfEdges(); j++){
107        plan[i][j] = 0;
108      }
109    }
110
111 //     19. For i = lastTime to 1
112    for (i=papers.length -1 ; i>=0; i--) {
113
114
115 //       20.        IsAllZero = true
116      isAllZero = true;
117 //       21.        For j = 1 to numberOfEdge
118      for (j=0; j< papers[0].getNumberOfEdges(); j++) {
119 //       22.          If AngleDifference[j][i] = -1,
120        if (angleDifference[j][i] == -1){
121 //       23.                    Do Plan[Phase][j] = -1
122 //       24.                    isBuildingPlan = true
123 //       25.                    IsAllZero = false
124          plan[phase][j] = -1;
125          isBuildingPlan = true;
126          isAllZero = false;
127        }
128 //       26.              If AngleDifference[j][i] = 1,
```

```
129             if (angleDifference[j][i] == 1){
130 //              27.                         Do Plan[Phase][j] = 1
131 //              28.                         isBuildingPlan = true
132 //              29.                         IsAllZero = false
133                 plan[phase][j] =  1;
134                 isBuildingPlan = true;
135                 isAllZero = false;
136               }
137             }
138 //          30.         If isBuildingPlan = true and isAllzero = true
139             if (isBuildingPlan == true && isAllZero == true) {
140 //              31.             Do lastPhase = Phase
141 //              32.             Phase++
142 //              33.             isBuildingPlan = false
143               phase++;
144               numberOfPhases  = phase;
145               isBuildingPlan = false;
146             }
147           }
148
149           int temp_plan[][] = new int[numberOfPhases][papers[0].getNumberOfEdges
                  ()];
150           for (i = 0; i < numberOfPhases; i++) {
151             for(j=0; j < papers[0].getNumberOfEdges() ; j++){
152               temp_plan[i][j] = plan[i][j];
153             }
154           }
155
156           // build edge table
157           float edgeTable[][] = new float[papers[0].getNumberOfEdges()][4];
158           for (i=0 ; i < papers[0].getNumberOfEdges(); i++) {
159             edgeTable[i][0] = (float) papers[0].getLine(i).getStartPoint().
                    getXOnPaper();
160             edgeTable[i][1] = (float) papers[0].getLine(i).getStartPoint().
                    getYOnPaper();
161             edgeTable[i][2] = (float) papers[0].getLine(i).getEndPoint().
                    getXOnPaper();
162             edgeTable[i][3] = (float) papers[0].getLine(i).getEndPoint().
                    getYOnPaper();
163           }
164
165           this.planForOrigami.setEdgeTable(edgeTable);
166           this.planForOrigami.setPlanTable(temp_plan);
167           this.planForOrigami.setNumberOfEdges(papers[0].getNumberOfEdges());
168           this.planForOrigami.setNumberOfPhases(numberOfPhases);
169         }
170
171         @Override
172         public void exportPlan(String fileName, Paper[] papers) {
173
174         }
175
176         @Override
177         public Plan getPlan() {
178           // TODO Auto-generated method stub
179           return this.planForOrigami;
180         }
181
182         @Override
183         public void build(Plan[] plans) {
184           // TODO Auto-generated method stub
185
186         }
187
188         @Override
189         public void exportPlan(String fileName) {
190           FilePlan filePlan = new FilePlanForOrigami();
191
192           filePlan.build(fileName, planForOrigami);
193         }
194       }
```

```
1   package com.drancom.programmableMatter.folding.simulator.
        simulatorForOrigamiWithStepFunction;
2
3   import com.drancom.programmableMatter.folding.controller.paper.Paper;
4   import com.drancom.programmableMatter.folding.dataFile.FileObj;
5   import com.drancom.programmableMatter.folding.origami.planner.Planner;
6   import com.drancom.programmableMatter.folding.origami.planner.
        PlannerForWiring;
7   import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
        .MainWindowForFoldingRobotOrigami;
8   import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
        .PlanForOrigami;
9   import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
        .PlanerForOrigami;
10  import com.drancom.programmableMatter.stepFunction.StepFunction;
11
12  public class SimulatorForOrigamiWithStepFunction {
13
14    /** /
15      public static final String PLAN_FILENAME = "V:\\com\\dran\\vc\\pm\\
            RigidOrigami006\\RigidOrigami\\save_8x8_s-shuttle\\
            plan_for_origami_s-shuttle.csv";
16      public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
            RigidOrigami006\\RigidOrigami\\save_8x8_s-shuttle\\m%05d.obj";
17      public static final int NUMBER_OF_FILES= 50;
18    /**/
19      public static final String PLAN_FILENAME = "V:\\com\\dran\\vc\\pm\\
            RigidOrigami006\\RigidOrigami\\save_8x8_hat\\
            plan_for_origami_8x8_hat.csv";
20      public static final String FILENAME = "V:\\com\\dran\\vc\\pm\\
            RigidOrigami006\\RigidOrigami\\save_8x8_hat\\m%05d.obj";
21      public static final int NUMBER_OF_FILES= 28;
22
23    /** /
24      public static final String PLAN_FILENAME = "c:\\foldingdata\\
            save_8x8airplain\\plan_for_origami_8x8airplain.csv";
25      public static final String FILENAME = "c:\\foldingdata\\
            save_8x8airplain\\m%05d.obj";
26      public static final int NUMBER_OF_FILES= 50;
27
28    /** /
29      public static final String PLAN_FILENAME = "c:\\foldingdata\\
            save_8x8box\\plan_for_origami_8x8box.csv";
30      public static final String FILENAME = "c:\\foldingdata\\save_8x8box\\m
            %05d.obj";
31      public static final int NUMBER_OF_FILES= 70;
32
33    /** /
34      public static final String PLAN_FILENAME = "c:\\foldingdata\\
            save_8x8sailboat\\plan_for_origami_8x8sailboat.csv";
35      public static final String FILENAME = "c:\\foldingdata\\
            save_8x8sailboat\\m%05d.obj";
36      public static final int NUMBER_OF_FILES= 50;
37
38    /** /
39      public static final String PLAN_FILENAME = "c:\\foldingdata\\
            save_8x8bench\\plan_for_origami_save_8x8bench.csv";
40      public static final String FILENAME = "c:\\foldingdata\\save_8x8bench\\
            m%05d.obj";
41      public static final int NUMBER_OF_FILES= 70;
42
43    /** /
44      public static final String PLAN_FILENAME = "c:\\foldingdata\\
            save_8x8table\\plan_for_origami_save_8x8table.csv";
45      public static final String FILENAME = "c:\\foldingdata\\save_8x8table\\
            m%05d.obj";
```

```
46      public static final int NUMBER_OF_FILES= 30;
47
48    /** /
49      public static final String PLAN_FILENAME = "c:\\foldingdata\\
            save_8x8table_b\\plan_for_origami_save_8x8table_b.csv";
50      public static final String FILENAME = "c:\\foldingdata\\save_8x8table_b
            \\m%05d.obj";
51      public static final int NUMBER_OF_FILES= 25;
52
53    /**/
54
55  //      public static final String FILENAME = "c:\\foldingdata\\save_box\\
          m00070.obj";
56  //      public static final int NUMBER_OF_FILES= 1;
57
58      Paper[] papers;
59      FileObj[] fileObjs;
60      MainWindowForFoldingRobotOrigami mainWindow;
61      public SimulatorForOrigamiWithStepFunction() {
62
63      }
64
65      void run() {
66        int i;
67        String fileName;
68
69        // Initiation
70        papers = new Paper[NUMBER_OF_FILES];
71        fileObjs = new FileObj[NUMBER_OF_FILES];
72        mainWindow = new MainWindowForFoldingRobotOrigami();
73
74
75        // load
76        for (i=0; i < NUMBER_OF_FILES; i++) {
77          fileName = String.format(FILENAME, i+1);
78          papers[i] = new Paper();
79          fileObjs[i] = new FileObj();
80          fileObjs[i].load(fileName, papers[i]);
81          papers[i].sortLine();
82        }
83
84        for (i=0; i < papers[0].getNumberOfEdges(); i++) {
85          System.out.printf("%d, %f, %f, %f, %f\n" , i, papers[3].getLine(i).
                getStartPoint().getXOnPaper()
86              , papers[0].getLine(i).getStartPoint().getYOnPaper()
87              , papers[0].getLine(i).getEndPoint().getXOnPaper()
88              , papers[0].getLine(i).getEndPoint().getYOnPaper());
89        }
90        // sort paper
91
92
93        Planner planer = new PlanerForOrigamiWithStepFunction(); // new
              planer
94
95        planer.build(papers);
96
97        planer.exportPlan(PLAN_FILENAME);
98
99        mainWindow.run(papers, (PlanForOrigami) planer.getPlan());
100
101     }
102
103     public static void main(String[] args) {
104       // TODO Auto-generated method stub
105       SimulatorForOrigamiWithStepFunction simulator = new
              SimulatorForOrigamiWithStepFunction ();
106       simulator.run();
107     }
108 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.
       simulatorForPlanOfOrigami;
2
3  import com.drancom.programmableMatter.folding.controller.paper.Line;
4  import com.drancom.programmableMatter.folding.controller.paper.Paper;
5  import com.drancom.programmableMatter.folding.controller.paper.Point;
6  import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
       .PlanForOrigami;
7  import com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamiWithStepFunction.PlanerForOrigamiWithStepFunction;
8
9  public class FilterOfPlanOfAlgorithm {
10   public boolean filtering(Paper paper, PlanForOrigami plan) {
11     if (paper.getNumberOfEdges() != plan.getNumberOfEdges()) {
12       return false;
13     }
14
15     int i, j, k;
16     Line line;
17     Point startPoint = new Point();
18     Point endPoint   = new Point();
19
20     float edgeTable [][] ;
21     int planTable [][];
22
23     edgeTable = plan.getEdgeTable();
24     planTable = plan.getPlanTable();
25     int numberOfLine = 0;
26     boolean isAllZero;
27     for (i=0; i< edgeTable.length; i++) {
28       isAllZero = true;
29       for (j = 0; j < planTable.length; j++) {
30         if (planTable[j][i] != 0 ) {
31           isAllZero = false;
32         }
33       }
34       if (isAllZero) {
35         startPoint = paper.getPoint(edgeTable[i][0], edgeTable[i][1], 0.0f)
               ;
36         endPoint = paper.getPoint(edgeTable[i][2], edgeTable[i][3], 0.0f);
37
38         line = paper.getLine(startPoint, endPoint);
39         line.setAngle(0.0f);
40       }
41
42     }
43
44
45
46     return true;
47   }
48 }
```

```
1  package com.drancom.programmableMatter.folding.simulator.
       simulatorForPlanOfOrigami;
2
3  import com.drancom.programmableMatter.folding.controller.paper.Paper;
4  import com.drancom.programmableMatter.folding.controller.paper.util.Vector;
5  import com.drancom.programmableMatter.folding.dataFile.FileObj;
6  import com.drancom.programmableMatter.folding.monitor.MainWindow;
7  import com.drancom.programmableMatter.folding.origami.planner.Planner;
8  import com.drancom.programmableMatter.folding.origami.planner.
       PlannerForWiring;
9  import com.drancom.programmableMatter.folding.simulator.Simulator;
10 import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
       .PlanForOrigami;
11 import com.drancom.programmableMatter.folding.simulator.
       simulatorForOrigamiWithStepFunction.PlanerForOrigamiWithStepFunction;
12
```

```
13 public class SimulatorOfPlanOfOrigami {
14
15   /** /
16     public static final String PLAN_FILENAME = "c:\\foldingdata\\
           save_airplain\\plan_airplain.csv";
17     public static final String FILENAME = "c:\\foldingdata\\save_airplain\\
           m%05d.obj";
18     public static final int NUMBER_OF_FILES= 50;
19
20   /**/
21     public static final String PLAN_FILENAME = "c:\\foldingdata\\save_box\\
           plan_box.csv";
22     public static final String FILENAME = "c:\\foldingdata\\save_box\\m%05d
           .obj";
23     public static final int NUMBER_OF_FILES= 70;
24
25   /** /
26     public static final String PLAN_FILENAME = "c:\\foldingdata\\
           save_sailboat2\\plan_sailboat2.csv";
27     public static final String FILENAME = "c:\\foldingdata\\save_sailboat2
           \\m%05d.obj";
28     public static final int NUMBER_OF_FILES= 95;
29
30   /** /
31     public static final String PLAN_FILENAME = "c:\\foldingdata\\
           save_8x8bench\\plan_save_bench.csv";
32     public static final String FILENAME = "c:\\foldingdata\\save_8x8bench\\
           m%05d.obj";
33     public static final int NUMBER_OF_FILES= 70;
34
35   /**/
36
37 //       public static final String FILENAME = "c:\\foldingdata\\save_box\\
       m00070.obj";
38 //       public static final int NUMBER_OF_FILES= 1;
39
40     Paper[] papersWithPlan;
41     Paper[] papersOriginal;
42     FileObj[] fileObjs;
43     MainWindow mainWindow;
44     public SimulatorOfPlanOfOrigami() {
45     }
46
47     void run() {
48       int i;
49       String fileName;
50
51       // init
52       papersWithPlan = new Paper[NUMBER_OF_FILES];
53       papersOriginal = new Paper[NUMBER_OF_FILES];
54       fileObjs = new FileObj[NUMBER_OF_FILES];
55       mainWindow = new MainWindow();
56
57       // load
58       for (i=0; i < NUMBER_OF_FILES; i++) {
59         fileName = String.format(FILENAME, i+1);
60         papersOriginal[i] = new Paper();
61         papersWithPlan[i] = new Paper();
62         fileObjs[i] = new FileObj();
63         fileObjs[i].load(fileName, papersOriginal[i]);
64         fileObjs[i].load(fileName, papersWithPlan[i]);
65         }
66
67
68
69       Planner planer = new PlanerForOrigamiWithStepFunction();
70
71       planer.build(papersOriginal);
72
73
74       FilterOfPlanOfAlgorithm filterOfPlanOfAlgorithm = new
             FilterOfPlanOfAlgorithm();
75
76       for (i=0; i < NUMBER_OF_FILES; i++) {
77         filterOfPlanOfAlgorithm.filtering(papersWithPlan[i], (
               PlanForOrigami) planer.getPlan());
78       }
79
80       System.out.print(getAveDistanceDifference(papersOriginal,
             papersWithPlan));
81
82
83
84
85       planer.exportPlan(PLAN_FILENAME, papersOriginal);
86
87       mainWindow.run(papersOriginal);
88
89     }
90
91     protected float getAveDistanceDifference(Paper[] papers0, Paper[]
           papers1){
92
93       float aveDistanceDifference;
94
95       int i, j;
96
97       Vector v0, v1;
98       aveDistanceDifference = 0.0f;
99       for( i = 0; i<papers0.length; i++) {
100        for( j =0 ; j< papers0[i].getNumberOfPoints(); j++) {
101
102          v0 = papers0[i].getPoint(j).getVectorInReal();
103          v1 = papers1[i].getPoint(j).getVectorInReal();
104
105          v1.invert();
106          v0.addVector(v1);
107
108          aveDistanceDifference += (float) (Math.sqrt( v0.getX()*v0.getX()
109            + v0.getY()*v0.getY()
110            + v0.getZ()*v0.getZ() ) ) ;
111
112        }
113      }
114
115      System.out.print(aveDistanceDifference );
116      aveDistanceDifference /=  ((double) papers0[0].getNumberOfPoints());
117
118
119
120      return aveDistanceDifference;
121    }
122
123    public static void main(String[] args) {
124      // TODO Auto-generated method stub
125      SimulatorOfPlanOfOrigami simulator = new SimulatorOfPlanOfOrigami();
126      simulator.run();
127    }
128 }
```

```
1 package com.drancom.programmableMatter.stepFunction;
2
3 public class StepFunction {
4   public static float[] stepFunction (float [] P, float eps){
5 //     StepFtn[P_(* list of y-
6 //           value at each time step--starting at time 1 for convenience*),
7 //           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)l / (
              float)x ) * (float)i;
          points[m][l][1] = POINTS[k][l][1] / (float) y + ( (float)l / (
              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"
```

```
21
22      };
23
24    public static final int NUMBER_OF_PLAN_FILES= 1;
25
26    public static final String PLAN_FOR_ORIGAMIS_FILENAME = "c:\\foldingdata
          \\save_plan\\%splan_for_origamis %d.%s";
27    public static final String PLAN_FOR_ORIGAMIS_FILETYPE = "csv";
28
29
30    void run() {
31      int i;
32      int j;
33      int k;
34  /**/
35      FileObj[] fileObjs;
36      PlanForOrigami[] plansForOrigami;
37      PlanForOrigami[] inputPlansForOrigami;
38      MonitorOfPlanGroupOfPlanForOrigamis monitor;
39
40      Planner [] planers;
41
42      int numberOfPlansForOrigamis;
43      int optimalPlanForOrigamis;
44      int numberOfActiveEdgeOfOptimalPlanForOrigamis;
45
46      int optimalGroupsForOrigamis;
47      int numberOfGroupsOfOptimalNumberOfGroups;
48
49      String tempString;
50
51      // Initiation
52      fileObjs =  new FileObj[NUMBER_OF_PLAN_FILES];
53      plansForOrigami = new PlanForOrigami[NUMBER_OF_PLAN_FILES];
54
55      // loads
56
57      for (i=0; i < NUMBER_OF_PLAN_FILES; i++) {
58          plansForOrigami[i] = new PlanForOrigami();
59          plansForOrigami[i].load(PLAN_FILENAME[i]);
60      }
61
62      ProgrammableMatter pm = new ProgrammableMatter(4, 4);
63      pm .run(plansForOrigami);
64
65    }
66
67    public static void main(String[] args) {
68      SimulatorForTileProgrammableMatter simulator = new
            SimulatorForTileProgrammableMatter();
69      simulator.run();
70    }
71
72  }
```

```
1  package com.drancom.programmableMatter.tile.clipSMA;
2
3  public class Tile {
4    final static boolean SMA  = true;
5    final static boolean NO_SMA = false;
6
7    boolean typeOfTile;
8    float points[][] = new float[3][2];
9
10   int edgeSwitchs[][] = new int[3][2];
11
12   Tile (boolean typeOfTile, float [][] points, int[][] edgeSwitchs) {
13     int i,j;
14     this.typeOfTile = typeOfTile;
15     for (i=0; i<points.length; i++) {
16       for (j=0; j < points[i].length; j++) {
17         this.points[i][j] = points[i][j];
18       }
19     }
20
21     for (i=0; i<edgeSwitchs.length; i++) {
22       for (j=0; j < edgeSwitchs[i].length; j++) {
23         this.edgeSwitchs[i][j] = edgeSwitchs[i][j];
24       }
25     }
26   }
27
28   boolean isSMA(){
29     return (typeOfTile==SMA);
30   }
31
32   boolean isNO_SMA(){
33     return (typeOfTile==NO_SMA);
34   }
35
36   boolean getTypeOfTile() {
37     return typeOfTile;
38   }
39
40   int hasTwoPoints(float pointX0, float pointY0, float pointX1, float
         pointY1){
41   // return edge enumber
42
43     int i;
44
45     for (i=0; i<3; i++) {
46       if ((points[i][0] == pointX0 && points[i][1] == pointY0
47           && points[(i+1)%3][0] == pointX1 && points[(i+1)%3][1] == pointY1
                )
48           ||
49         (points[i][0] == pointX1 && points[i][1] == pointY1
50           && points[(i+1)%3][0] == pointX0 && points[(i+1)%3][1] == pointY0
                )) {
51         return i;
52       }
53     }
54     return -1;
55   }
56
57   boolean hasPoint(float pointX0, float pointY0) {
58     int i;
59     for (i=0; i<3; i++) {
60       if (points[i][0] == pointX0 && points[i][1] == pointY0) {
61         return true;
62       }
63     }
64     return false;
65   }
66
67   boolean setEdgeSwitch(boolean typeOfTile
68           , int edgeId
69           , int switchId
70           , int typeOfEdge){
71     int i;
72
73     if (this.typeOfTile == typeOfTile && edgeId < 3 && switchId < 2) {
74       edgeSwitchs [edgeId][switchId] = typeOfEdge;
75       return true;
76     }
77
78     return false;
79   }
80   String getTileCode () {
```

```
81
82      String power0 = new String();
83      String power1 = new String();
84      int code0 = 0;
85      int code1 = 0;
86      power0 += Integer.toString(edgeSwitchs[0][0]);
87      power0 += Integer.toString(edgeSwitchs[1][0]);
88      power0 += Integer.toString(edgeSwitchs[2][0]);
89
90      power1 += Integer.toString(edgeSwitchs[0][1]);
91      power1 += Integer.toString(edgeSwitchs[1][1]);
92      power1 += Integer.toString(edgeSwitchs[2][1]);
93
94      if (edgeSwitchs[0][0]!=0) {
95        code0 += 1;
96      }
97      if (edgeSwitchs[1][0]!=0) {
98        code0 += 2;
99      }
100     if (edgeSwitchs[2][0]!=0) {
101       code0 += 4;
102     }
103
104     if (edgeSwitchs[0][1]!=0) {
105       code1 += 1;
106     }
107     if (edgeSwitchs[1][1]!=0) {
108       code1 += 2;
109     }
110     if (edgeSwitchs[2][1]!=0) {
111       code1 += 4;
112     }
113
114     return power0 + " " + power1 + " " + Integer.toString(code0) + Integer.
          toString(code1) ;
115   }
116 }
```

```
1  package com.drancom.programmableMatter.tile.springSma;
2
3  import com.drancom.programmableMatter.folding.simulator.simulatorForOrigami
       .PlanForOrigami;
4
5  public class ProgrammableMatter {
6    final static float POINTS[][][] =
7        {{{0.0f, 0.0f}, {0.0f, 0.5f}, {0.5f, 0.5f}},
8        {{0.5f, 0.5f}, {0.5f, 0.0f}, {0.0f, 0.0f}},
9        {{1.0f, 0.0f}, {0.5f, 0.0f}, {0.5f, 0.0f}},
10       {{0.5f, 0.5f}, {1.0f, 0.5f}, {1.0f, 0.0f}},
11       {{1.0f, 1.0f}, {1.0f, 0.5f}, {0.5f, 0.5f}},
12       {{0.5f, 0.5f}, {0.5f, 1.0f}, {1.0f, 1.0f}},
13       {{0.0f, 1.0f}, {0.5f, 1.0f}, {0.5f, 0.5f}},
14       {{0.5f, 0.5f}, {0.0f, 0.5f}, {0.0f, 1.0f}}};
15
16   final static boolean EDGESWITCH [][] =
17     {{false, false}, {false, false}, {false, false}};
18
19   Tile tiles[];
20
21   ProgrammableMatter(int x, int y){
22     int i, j, k, l, m;
23     float[][][] points = new float [8*x*y][3][2];
24
25     m=0;
26     for ( i = 0 ; i< x ; i++) {
27       for (j = 0; j < y ; j++) {
28         for (k = 0; k < 8 ; k++) {
29           for (l = 0; l < 3 ; l++) {
30
```

```
31             points[m][l][0] = POINTS[k][l][0] / (float) x + ( (float)i / (
                   float)x ) * (float)i;
32             points[m][l][1] = POINTS[k][l][1] / (float) y + ( (float)j / (
                   float)y ) * (float)j;
33
34
35           }
36           m++;
37         }
38       }
39     }
40
41     tiles = new Tile[8 * x * y];
42     for ( i = 0 ; i < 8 * x * y ; i += 2 ) {
43       tiles[i] = new Tile(Tile.MOUNT, points[i], EDGESWITCH);
44       tiles[i + 1] = new Tile(Tile.VALLEY, points[i+1], EDGESWITCH);
45     }
46   }
47
48   void setTileSwitch(float pointX0
49             , float pointY0
50             , float pointX1
51             , float pointY1
52             , boolean typeOfTile
53             , int switchId
54             , boolean isTurnOn){
55     int i;
56     int edgeId;
57
58     for (i=0; i<tiles.length; i++) {
59       edgeId = tiles[i].hasTwoPoints(pointX0, pointY0, pointX1, pointY1);
60       if (tiles[i].getTypeOfTile() == typeOfTile
61           && edgeId != -1){
62         tiles[i].setEdgeSwitch(typeOfTile, edgeId, switchId, isTurnOn);
63       }
64     }
65   }
66
67   void printCodeOfTiles(){
68     int i, j;
69     j = 0;
70     for (i = 0; i < tiles.length; i++) {
71       System.out.printf("%d ", tiles[i].getTileCode());
72       if (j < 7) {
73         j++;
74       } else {
75         j=0;
76         System.out.printf("\n ");
77       }
78     }
79   }
80
81   void run(PlanForOrigami[] planForOrigami) {
82     int i, j;
83     int numberOfEdges;
84     int numberOfPhases;
85
86     for (i = 0; i < 1; i++) {
87       int [][] planTable = planForOrigami[i].getPlanTable();
88       float [][] edgeTable = planForOrigami[i].getEdgeTable();
89
90       numberOfEdges = planForOrigami[i].getNumberOfEdges();
91       numberOfPhases = planForOrigami[i].getNumberOfPhases();
92
93       for(j = 0 ; j < numberOfEdges; j++){
94         if (planTable[0][j] == 1){
95           setTileSwitch(edgeTable [j][0]
96                 , edgeTable [j][1]
97                 , 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 power1, power0;
    power1 = 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]) {
      power1 += 1;
    }
    if (edgeSwitchs[1][1]) {
      power1 += 2;
    }
    if (edgeSwitchs[2][1]) {
      power1 += 4;
    }
    return power1 * 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;
```

```java
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();
  }
}
```

```java
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);
```

```
73          }
74  /**/
75          if (minEngPaper.getGlobalEnergy()== 0
76              || (paperArray.size()>3 && (paperArray.get(paperArray.size()-1).
                    getGlobalEnergy()
77              == paperArray.get(paperArray.size()-3).getGlobalEnergy() ))) {
78            break;
79          }
80      }
81
82      for(i=0 ; i < paperArray.size(); i++) {
83        System.out.println(paperArray.get(i).getGlobalEnergy());
84      }
85
86      return paperArray;
87    }
88  }
```

```
1  package com.drancom.programmableMatter.unfolding.localEnergyFunction;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.Hashtable;
6  import java.util.SortedMap;
7
8  import com.drancom.programmableMatter.folding.controller.paper.Paper;
9  import com.drancom.programmableMatter.folding.controller.paper.Point;
10 import com.drancom.programmableMatter.folding.controller.paper.Polygon;
11 import com.drancom.programmableMatter.folding.monitor.MonitorOfPaperArray;
12 import com.drancom.programmableMatter.unfolding.unfolder.Unfolder;
13
14 public class UnfolderWithLocalEnergyFunction implements Unfolder{
15
16
17   @Override
18   public ArrayList<Paper> unfolding(Paper paper) {
19     /**
20      *
21      * 1. try unfolding all of the lines has edges Type A: Unfolding each
22      * line as much as possible. Type B: Unfolding each line until E became
23      * the smallest. 2. pick the paper has smallest energy 3. record array.
24      * 4. repeat 1 and 2 until all of the lines are unfolded.
25      */
26     int i, j, k;
27
28     ArrayList<Paper> paperArray = new ArrayList<Paper>();
29
30     Paper minEngPaper;
31     Paper unfoldingPaper;
32     Paper workingPaper;
33     Polygon polygon;
34     Point[] points;
35
36     float [] listOflocalEnergy       = new float[paper.getNumberOfEdges()];
37     float [] sortedListOflocalEnergy  = new float[paper.getNumberOfEdges()
           ];
38     int   [] listOfIndexOfLocalEnergy = new int[paper.getNumberOfEdges()];
39
40     // minEngPaper
41     unfoldingPaper = paper.snapshot();
42     minEngPaper = unfoldingPaper.snapshot();
43
44     paperArray.add(unfoldingPaper.snapshot());
45
46
47
48     for (k = 0; k < paper.getNumberOfEdges(); k++) {
49
50       // get a list of lines sorted by energy function
```

```
51
52       workingPaper = unfoldingPaper.snapshot();
53
54       for (i=0; i<paper.getNumberOfEdges(); i++) {
55
56         listOflocalEnergy[i]     = workingPaper.getLocalEnergy(workingPaper.
               getLine(i));
57         sortedListOflocalEnergy[i]  = listOflocalEnergy[i];
58       }
59       Arrays.sort(sortedListOflocalEnergy);
60       for (i=paper.getNumberOfEdges() - 1; i <= 0; i--) {
61         for (j=0; j<=paper.getNumberOfEdges() ; j++ ) {
62           if (sortedListOflocalEnergy[i] == listOflocalEnergy[j]) {
63             listOfIndexOfLocalEnergy[(paper.getNumberOfEdges() - 1) - i] =
                   j;
64             break;
65           }
66         }
67       }
68
69       for (i = 0; i<workingPaper.getNumberOfEdges(); i++ ) {
70         if (workingPaper.changeAngleForSmallestGlobalEnergy(
71           workingPaper.getLine(listOfIndexOfLocalEnergy[i]))) {
72
73           // compare minEnergy Paper and paper
74           if (minEngPaper.getGlobalEnergy() > workingPaper
75             .getGlobalEnergy()) {
76             minEngPaper = workingPaper;
77           }
78               // unfolded
79           if (workingPaper.getGlobalEnergy() == 0.0f) {
80             paperArray.add(minEngPaper);
81             if (minEngPaper.getGlobalEnergy() == 0.0f) {
82               paperArray.add(0, paper.snapshot());
83               return paperArray;
84             }
85           }
86         }
87       }
88
89       unfoldingPaper = minEngPaper;
90       paperArray.add(minEngPaper);
91       System.out.println(minEngPaper.getGlobalEnergy());
92 /**/
93       if (minEngPaper.getGlobalEnergy() <= Paper.ERROR_RATIO_FOR_LANGTH ) {
94         MonitorOfPaperArray monitor = new MonitorOfPaperArray();
95         monitor.run(paperArray);
96       }
97 /**/
98       if (minEngPaper.getGlobalEnergy()== 0
99           || (paperArray.size()>3 && (paperArray.get(paperArray.size()-1).
                 getGlobalEnergy()
100          == paperArray.get(paperArray.size()-3).getGlobalEnergy() ))) {
101        break;
102      }
103
104
105      for(i=0 ; i < paperArray.size(); i++) {
106        System.out.println(paperArray.get(i).getGlobalEnergy());
107      }
108
109      return paperArray;
110    }
111    return paperArray;
112
113  }
114
115 }
```

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.