Technology
Arts Sciences
TH Köln

Origami Diagramming

Development of a desktop application for creating origami diagrams

Practical Project

by

Julian Hardtung

submitted to obtain the degree of

Bachelor of Science (B. Sc.)

at

TH Köln University of Applied Sciences
Campus Gummersbach
Institute of Informatics and Engineering

Course of Studies

Media Informatics

First supervisor: Prof. Dr. Martin Eisemann
TH Köln University of Applied Sciences

Second supervisor: Matthias Groß
TH Köln University of Applied Sciences

Gummersbach, March 1, 2020

Adresses: Julian Hardtung
Lachtstraße 12
51645 Gummersbach
ju.hardtung@gmx.de

Prof. Dr. Martin Eisemann
TH Köln University of Applied Sciences
Institute of Informatics and Engineering
Steinmüllerallee 1
51643 Gummersbach
martin.eisemann@th-koeln.de

Matthias Groß
TH Köln University of Applied Sciences
Institute of Informatics and Engineering
Steinmüllerallee 1
51643 Gummersbach
matthias.gross2@th-koeln.de

Contents

1 Introduction

Creating instructions for folding Origami models is a tedious and time consuming task. These so called origami diagrams (see Fig **??**) have to be accurate representations of the paper for every folding step, in order to unambiguously explain how to fold the model. Every flap, crease and edge has to be drawn (see Section 2.1 for exceptions), which makes the process slow and especially error-prone for complex models.

Historically, diagrams got either drawn by hand or created with the help of digital vector programs like Inkscape [1] or Adobe Illustrator [2]. Even though the digital diagramming improved the accuracy of the finished instructions, the task itself was still quite time consuming [**?**].

Up until the point of this publication there is no publicly available program that offers specific features for the origami diagramming process. The previously mentioned vector programs can be used for diagramming, however as they weren't developed with origami diagramming in mind, there are a lot of shortcomings in their feature set. These shortcomings of current programs will be further elaborated on in Section 3.

The goal for this project is to develop a desktop application that implements features specifically for the origami diagramming process. The standardized symbols and overall notations have to be included and the program has to offer functions that increase the efficiency of creating diagrams.

This work starts with an overview and subsequently a categorization of current origami diagramming notations. Based on these findings, a requirements analysis can be carried out in order to define the required features of the planned program. Any problems or other findings during the development process will be documented so that they may be beneficial to others in the future.

[1] https://inkscape.org/

[2] https://www.adobe.com/de/products/illustrator.html

2 Diagramming Notation

In order to define the concrete requirements of the planned diagramming program, all commonly used diagramming symbols and conventions have to be collected and categorized. After that groundwork a plan can be established on how to implement these findings in a desktop application.
The very fundamentals of Origami diagramming were developed and proposed by Akira Yoshizawa in his book *Atarashi Origami Geijutsu (New Origami Art)*[2] in 1954, which introduced a system of folding notation. Yoshizawas diagramming system is still widely used today and after refining by Samuel Randlett and Robert Harbin most commonly known as the *Yoshizawa–Randlett system*.
Despite its high popularity, different nuances and slight changes were made by different origami diagrammers over the years. To avoid further confusion, especially for beginners, american physicist and origami artist Robert J. Lang compiled notations that were in use by different folders and proposed a standard for all different folding sequences. In order to support his claims, Lang sent "a questionnaire to 25 diagrammers around the world"[1] and tried to coherently argue in favour of, or against the various results.
Langs efforts were made under the assumption, that "[...] unless there is pressing reason otherwise, we should use the standard notation developed by Yoshizawa" [1].

To start off we have to define what exact components a diagram consists of. Most importantly, there always is a visual representation of the paper in the current step. This representation should show all or most creases, flaps, edges and layers to accurately show how the actual paper model would look like (see Section 2.1 on exceptions).
Secondly, there has to be a description of the actual folding sequence for one step. This description is comprised of a textual explanation and a visual display of the step with the diagramming symbols (Section 2.2). Both the verbal and the visual instructions should be able to stand for themselves, although that might not always be possible for complex steps. More detailed rules and specific terms for the verbal instructions are described on Section 2.1.

2.1 General Diagramming Rules

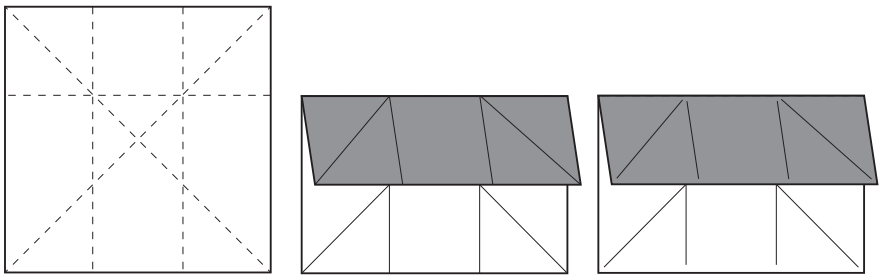
The following rules are general factors to keep in mind when creating origami diagrams. Some aspects here are not necessarily fulfilled by a specific feature within the planned program, but rather depend on the experience of the designer.

Be consistent
For consistency's sake the artist should stick with one notation within a diagram. One example would be the different ways to symbolize a Mountain Fold. These folds can either be drawn by a *dash dot* line or by a *dash dot dot* line.

Use the right origami grammar
As earlier established there should be an explanatory text for each folding step. But in order to precisely and clearly describe a fold a consistent grammar has to be used. An important rule is defined by R. Lang: "origami nouns are not hyphenated, but verbs are" [1]. This means that someone *reverse-folds* a flap into the interior but on the other hand flattens out a reverse fold. Furthermore one should capitalize only those origami moves that " [...] incorporate names, such as Elias-stretch (verb) or Elias stretch (noun) [...]" [1]. The common origami bases like the Bird Base, Waterbomb Base or others are to be capitalized entirely.

Distort the model to show all the layers
Slightly distorting the model to show otherwise hidden layers might be one of the most important general rules, as this one aspect hugely increases the readability and clarity of a diagram. Offering as many reference points as possible makes following a diagram easier and reduces confusion.

Creases should not contact the edges
Already existing creases from previous steps should be drawn with a thin, continuous line and **not** contact the edges of the paper (see Figure 2.1). The only exception is when the creases go under a layer of paper. In this case the line that represents the crease does touch the edge of the upper layer.



(a) Folds made (b) Wrong way (c) Correct way

Figure 2.1: Correct way to draw existing creases

Show one white and one colored side of the paper

When using different colored sides for the paper, the artist can create clearer diagrams. Especially once the instructions get more complex, every additional reference point is helpful.

Enlarging the model in a step

Enlarging the view on a model in a specific step can be accomplished by two methods. The first method mentioned by Lang is marking the area of focus with a circle before enlarging the view. This is especially helpful when focusing on smaller parts during shaping, such as the head. Alternatively the model can simply be enlarged, "because if you've drawn the diagrams neatly, it is obvious when the drawing has changed scale (and most people won't notice it, anyhow)" [1].

Show the result of complex folding steps

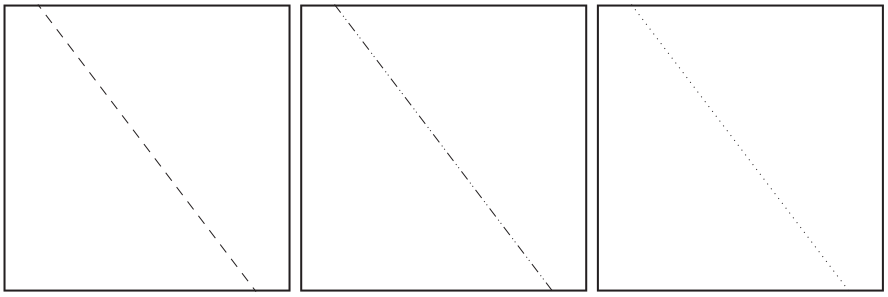
When giving complex folding instructions for a single step (especially when these folds get unfolded in the next step), one should always show the result of said folding sequence. Even though this approach may seem counterproductive as it increases the workload, but once again, the goal is to maximize clarity and remove any ambiguity when creating origami diagrams.

2.2 Diagramming Notation

As there are quite a few simple and self-explanatory symbols, there will only be a further explanation if there are any additional rules or things to keep in mind while using them. Furthermore, on occasion Lang gives multiple valid symbols or ways to visualize a certain fold or folding sequence. The goal in these cases should be to give the user a choice on what method to use for a diagram.

2.2.1 Folds

Starting with the very basics in origami, there has to be a differentiation between the types of folds. Generally, there are 2 different kinds, namely the Valley Fold and Mountain Fold, which are shown in Figure 2.2a and Figure 2.2b respectively. Additionally the so called X-Ray Line is shown in Figure 2.2c. Even though it is not a real fold, it was added here to show the three different forms a line can be drawn in origami diagrams. In order to indicate what kind of fold the X-Ray line is representing, you should extend the line past the paper and use a valley fold or mountain fold line.



(a) Valley Fold (b) Mountain Fold (c) X-Ray Line

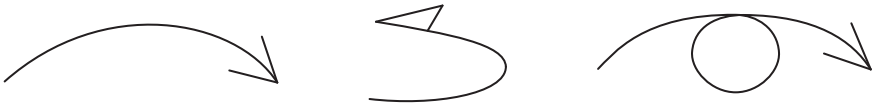
Figure 2.2: Different Lines in Origami Diagramming

Important for the displaying of lines is, that they don't start or end with a gap. This method ensures that there isn't any ambiguity on what reference points are needed for the fold.

2.2.2 Arrows

In order to show the folding steps unambiguously, there have to be arrows that indicate the direction the paper has to be folded. Showing only a valley or mountain fold leaves room for interpretation, which gets eliminated by

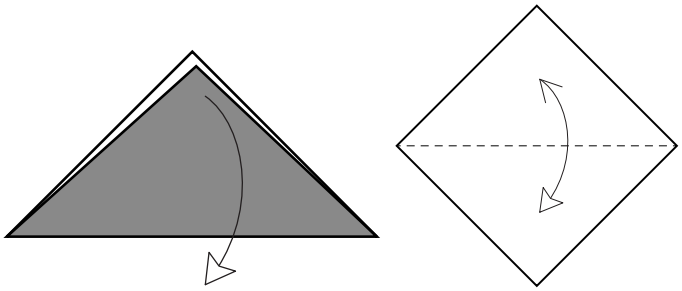
the addition of these specific arrows. In origami diagramming there are two groups, the Arrows of Motion and Arrows of Action. While the arrows of motion describe where the paper is folded to, the arrows of action indicate an action perfomed on the paper itself.



(a) Valley Fold (b) Mountain Fold (c) Turn the paper over

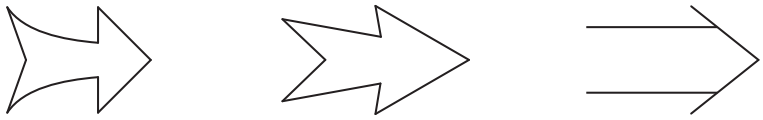
Figure 2.3: Arrows of Motion

The unfolding arrow can either stand alone and signalize an unfolding action, or be combined with a valley/mountain fold and indicate a folding and unfolding procedure.



(a) Unfold (b) Fold and unfold

Figure 2.4: Unfolding arrows



(a) Push here (b) Pull the paper out (c) Inflate here

Figure 2.5: Arrows of Action

2.2.3 Clarifying Diagrams

Leader

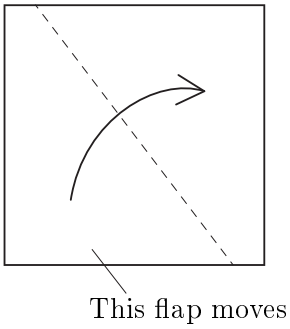


Figure 2.6: Use a leader to give additional information if needed

Equal distances

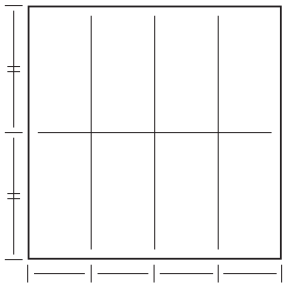


Figure 2.7: Equal Distances

Equal angles

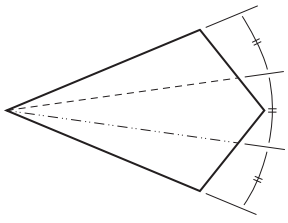


Figure 2.8: Equal Angles

Rotations

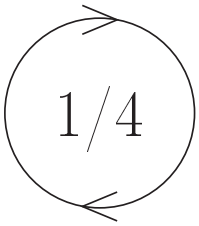


Figure 2.9: Rotation Symbol

X-Ray View

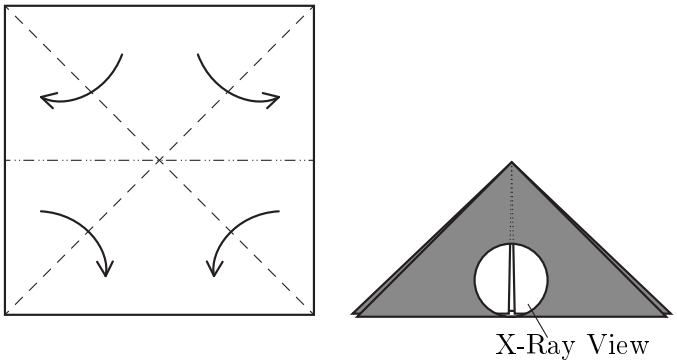


Figure 2.10: The X-Ray View shows hidden layers

Repetitions

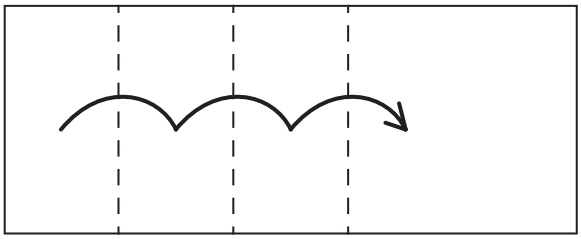


Figure 2.11: Fold over and over

For complicated repetitions you should show the result of the first fold that is to be repeated.
The repetition box from Figure 2.12 shows what exact steps are to be repeated and how many times.

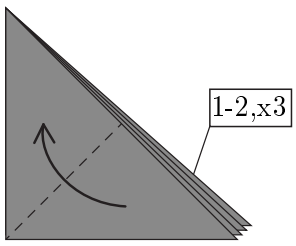


Figure 2.12: Repetition Box

Next view here

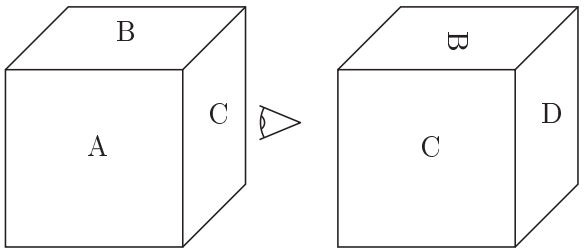


Figure 2.13: Next View Here

Hold here

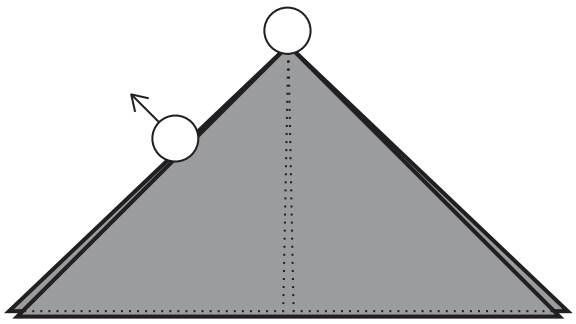
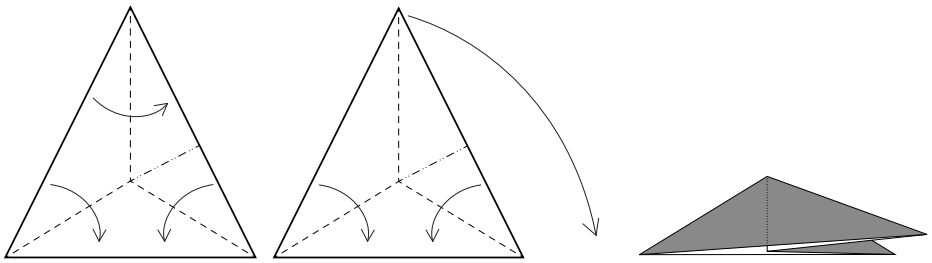


Figure 2.14: Hold here + hold here and pull

2.2.4 Folding Procedures

Rabbit Ear

Lang gives two valid methods to visualize a Rabbit Ear. While Figure 2.15a technically shows the more accurate format (according to the previously defined rules in Section 2.1), method B (see Figure 2.15b) is also desireable, as it shows the overall motion of the paper.

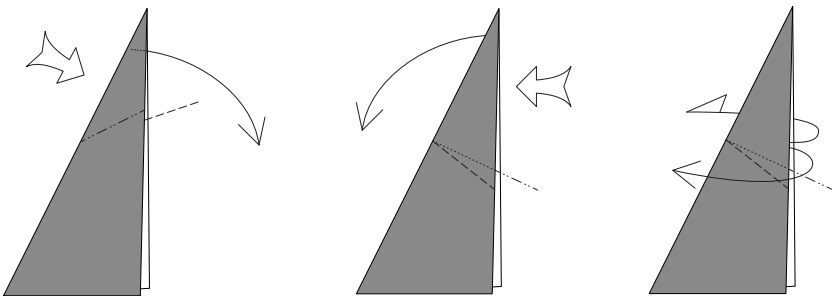


(a) Rabbit Ear A (b) Rabbit Ear B (c) Rabbit Ear Result

Figure 2.15: Both methods show a rabbit ear fold

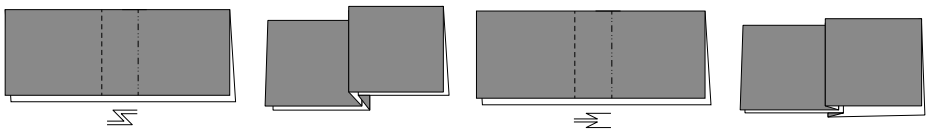
Reverse Folds

Although Lang proposes two valid notations for outside reverse folds, for consistency's sake Method A should be used (see Figure 2.16b).

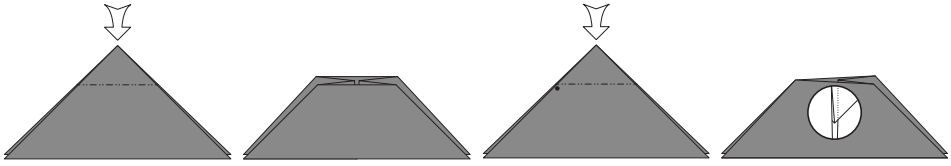


(a) Inside Reverse Fold (b) Outside Reverse Fold (c) Outside Reverse Fold

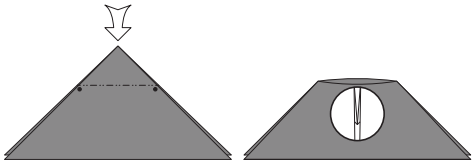
Figure 2.16: Different reverse folds



(a) Crimping (b) Pleating



(a) Open Sink (b) Mixed Sink



(c) Closed Sink

Crimping and Pleating

Sinks

3 Requirement Analysis

After collecting and categorizing all diagramming notations, the actual requirements of the planned system can be defined more precisely. On basis of the previous section (Section 2) three different requirement fields become apparent.

- Diagramming Symbols
- General Diagramming Rules
- General Program Tools

3.1 Diagramming Symbols

Beginning with the most important field, the diagramming symbols form the basis of diagramming and implementing all of them is imperative. Without all the different lines, arrows and other symbols there will be ambiguity in the later folding instructions. For a comprehensive list of all required diagramming symbols see Section 2.2.

3.2 General Diagramming Rules

The more abstract rules and best practices while creating origami diagrams already got established in Section 2.1. Including them in the system has to be decided on a case to case basis, as some rules have to be kept in mind by the artist during diagramming. Things like using the right *origami grammar* or *distorting the model in order to show otherwise hidden layers and edges* (see Section 2.1) rely heavily on the experience of the artist.
Though in the future some of these rules could be displayed as a hint within the program. And in the case of the right origami grammar, it could be thinkable to develop a system that automatically adds the description text of a step based on the symbols used.

3.3 General Program Tools

Outside of providing all the diagramming symbols and including the general diagramming rules where possible, the program has to offer common tools that have become standard in most vector programs.

- **Select, Move, Resize, Rotate:** editing an object (line, symbol etc.)
- **Delete:** delete diagramming symbol or whole step
- **Duplicate:** duplicate diagramming symbols or whole step
- **Copy & Paste:** copy or past an object (line, symbol etc.)
- **Undo & Redo:** undo or redo the last action (adding a line, symbol etc.)
- **Zoom, Rotate, Flip:** editing a step
- **Save, Open:** saving and opening an origami diagram
- **Export:** exporting a finished diagram to e.g .pdf

Even though these features are quite important for the later usability of the planned program, the focus was first set on implementing all origami specific tools first, as this provided the biggest unique feature set when comparing to already existing vector programs.

4 Program Features and Use

The structure of the developed origami diagramming program (ff. Origrammer) follows a similar approach to the real life diagramming process. To create a new model, we start with some general decisions. When selecting the "File > New" menu item, a new dialog box opens (see Figure 4.1) where *paper shape, size* and the *paper color* of both sides can be specified. Additionally, the text fields *Title, Author, Comments* can be filled out to provide further information of the model. These options can also be accessed and changed during the diagramming process (except for the paper shape) when selecting "Edit > Model Preferences".

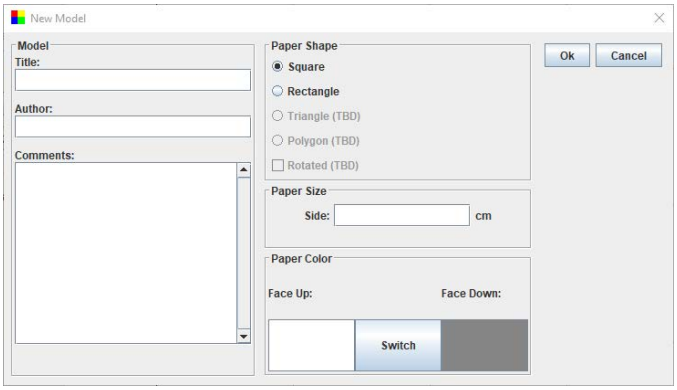


Figure 4.1: New Model dialog box

After specifying all the model settings, the main diagramming window is presented to the user (see Figure 4.3). This main editing area consists of four segments:

Side Panel

On the left side of the main window the side panel is located. It presents the toolbar where the user can switch between inputting lines, arrows, general origami symbols, the selection tool (further explained later), a measurement tool, as well as a filling tool. Furthermore, a helping grid can be displayed and adjusted to ease the input of lines. Additionally, when requiring more clarity during the diagramming process, the faces that got filled with the filling tool as well as all vertices can be turned invisible.

Top Panel

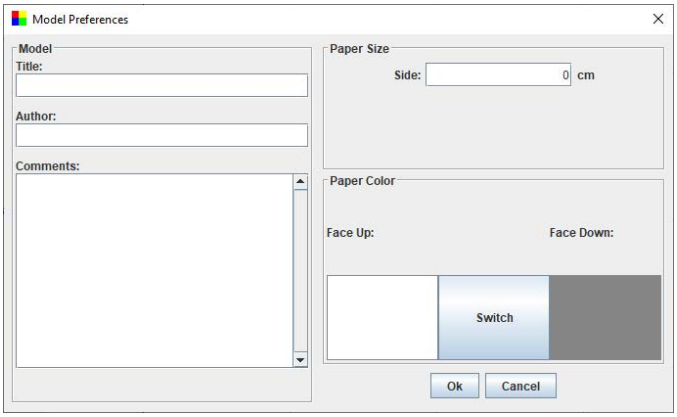


Figure 4.2: Model Preferences dialog box

The top panel is directly connected to the side panel, as it shows the actual settings, depending on what tool from the toolbar is active. Such as for the Line Input Tool, the top panel displays a combo box that contains the previously mentioned six different arrow types of origami. For other active tools this panel can show scaling or rotating options, different combo boxes, text fields or check boxes for a variety of adjustments.

Navigation Panel

Unlike the aforementioned panels, the navigation panel directly influences the model panel. As the name already states, this panel allows the navigation through the individual diagram steps. It also provides a text field to give the textual instructions for the current step. Furthermore different options can be selected on what should happen when a new step is created. The user can either create an empty step, copy the previous step or create a step with the basic paper shape that got defined in the *Model Preferences*.

Model Panel

The previously specified piece of paper is displayed in the model panel. This is the place where the paper, together with the arrows and other symbols, is rendered to represent the current state of the paper for each folding step.

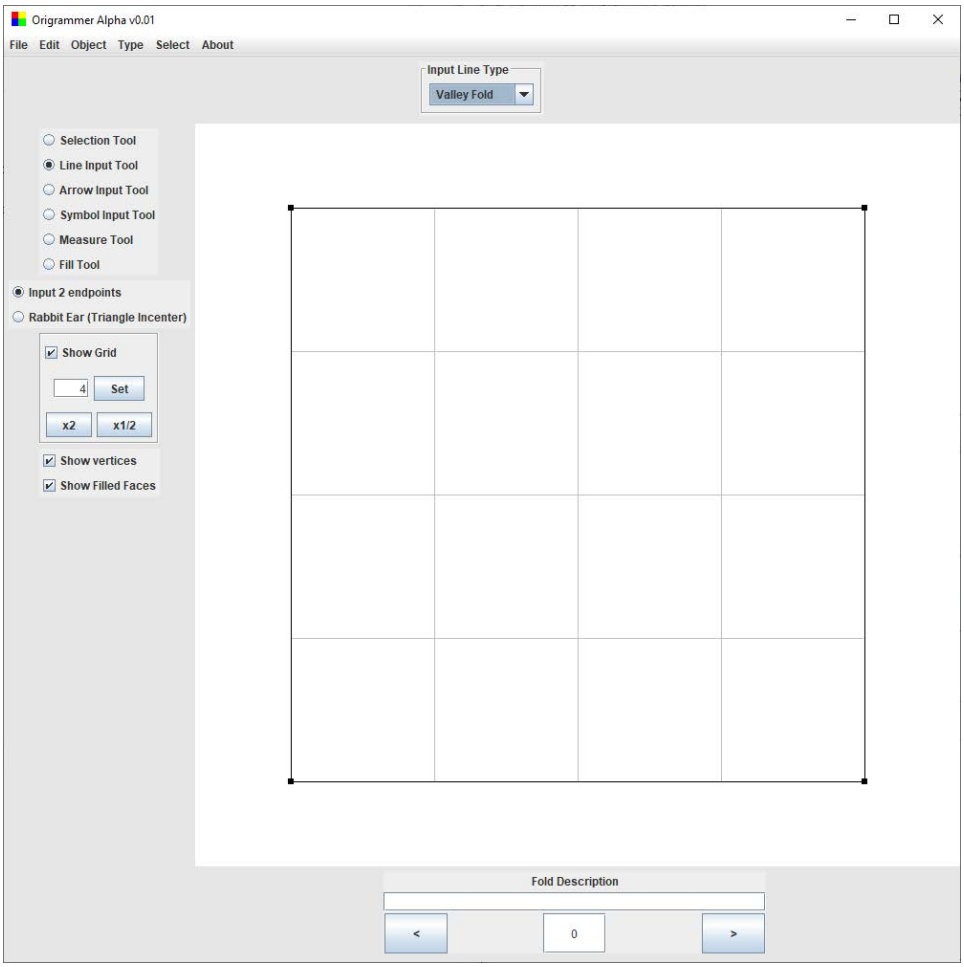


Figure 4.3: Origrammer main window

5 Problems & other Findings

This section discusses decisions and problems during the development process.
Rendering the lines of the paper, all the arrows and other symbols could be done in a plethora of ways. Although the use of graphics libraries like JavaFx, libgdx, lwjgl or jogl was one of the options, the Origrammer relies only on Graphics2D. The more sophisticated graphics libraries were mostly developed for game development and offer a lot of features that were not necessary for this project. Using only the general shapes of Graphics2D removes a lot of unnecessary complexity while offering a similar outcome.
To sum up, the Model Panel now consists of a JPanel on which a Graphics2D object is used to draw the diagram elements. But this decision also brought some restrictions and problems.

5.1 Displaying of Arrows/Symbols

As some arrows and symbols are quite complex it would have been preferable just to use images of the arrows/symbols and place them over the paper model (which only consists of simple lines). So the requirements were to display images with variable position, size and rotation. The most obvious solution was to place a JLabel for each arrow or symbol and use its `setIcon(BufferedImage img)` method to render a `BufferedImage img` of the arrows/symbols. Placing a JLabel on absolute x,y coordinates is done by using the `setBounds(x, y, width, height)` method with the mouse `x` and `y` coordinates and the picture `width` and `height`. Unfortunately there didn't seem to be a way to rotate a JLabel itself, so another way had to be found. The next idea was to rotate the BufferedImage itself and that did work quite well after calculating the new bounds of the JLabel based on the BufferedImage size.
While that solution did technically work, the outcome was not satisfactory as the image quality suffered a lot when scaling and zooming. Using vector graphics instead of .png files would avoid this quality loss, but natively there was no way to load a .svg file into the `Image` or `BufferedImage` class. To solve this problem the library Batik[3] was used. This allowed to load vector graphics of all arrows and symbols. Then after scaling and rotating, the

[3] https://xmlgraphics.apache.org/batik//

vector graphics can be rasterized (again, with Batik) in order to use them as ImageIcons of the previously mentioned labels. Unfortunately rotating the arrows and symbols turned out to be a problem now. As .svg files contain no explicit width or height values, calculating the updated bounds of the label after a rotation was not longer possible. As a result the rotated BufferedImage got cut off at the old bounds of its label. That is why the current implementation uses a unique vector graphic for every 22.5° rotation and loads them when necessary. The 22.5° angle was deliberately chosen as the paper in origami often gets folded at 90°, 45° or 22.5° angles (smaller and smaller bisectors with 22.5° as the smallest angle commonly used).

As a result the current implementation of using arrows and symbols works like this:

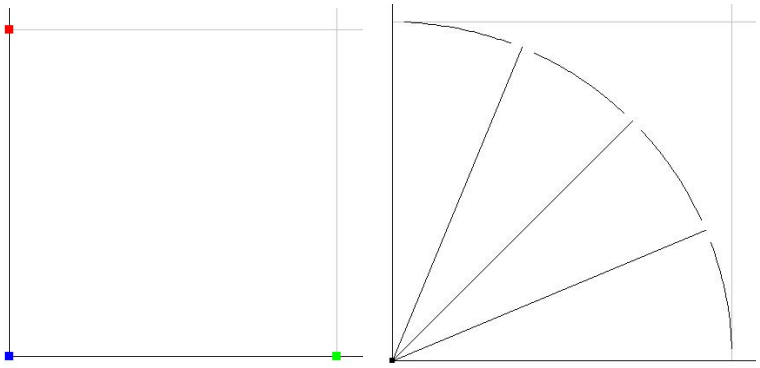
1. Load the .svg file of an arrow or symbol with the required rotation
2. Scale it as necessary
3. Rasterize it with Batik to a BufferedImage
4. Use this BufferedImage as an ImageIcon of a JLabel
5. Place this JLabel with `label.setBounds(x, y, width, height)` on the diagram (`width` and `height` are automatically determined by Batik when rasterizing).

This solution fulfills almost all requirements, as arrows and symbols can be placed freely by absolute coordinates and they can be rotated and scaled without quality loss. But there are still two problems.
Firstly, the bounding box of the JLabel is always square and therefore not accurate. This is another problem with the missing width and height values of .svg files. Though this could potentially be fixed by individually calculating the new x and y coordinates as well as the new width and height after scaling and rotating an arrow or symbol.
The second problem is unwanted scaling when rotating an arrow or symbol. This happens when a long image is rotated to 45°, 135°, 225° or 315° (so the image goes from one corner to another) as the JLabel tries to display the largest ImageIcon possible within its bounds. Again, both of these problems can be traced back to the missing width and height values of vector graphics.

The most flexible and accurate solution that solves every problem would still be either to rebuild all arrows and symbols by hand with the simple Shape objects inside Java or to transcode vector graphics into the Shapes. These two approaches could be used in the future but couldn't be tested yet as both solutions would take a considerable amount of time and rewriting of existing code.

5.2 Breaking Calculations

One problem in the current version of Origrammer is a possibly wrong user input. In case of the *Equal Angle* symbol the user has to specify 3 vertices on the diagram in a specific order. The first selected vertex is always the corner point (blue vertex on Fig. 5.1a), the second vertex has to lie vertically (the red vertex) and the last one has to lie horizontally (green vertex) towards the corner point. When using this order, the resulting symbol is drawn correctly for all four quadrants of a coordinate plane. Consequently the *Equal Angle* symbol can be drawn facing the top left, top right, bottom left or bottom right while also providing settings for a custom amount of dividers.



(a) Input (b) Result

Figure 5.1: Equal Angle Symbol

This reliance on correct user input could be reduced with a tooltip or small explaining text when selecting the input mode or by implementing a detailed input validation for each symbol. Though the focus while developing the Origrammer was set on increasing functionality over usability first. This approach allowed a quicker implementation of all required features in order to demonstrate how a finished diagramming program could look like.

5.3 Saving and Opening Files

Being able to save and open diagrams created with the Origrammer was imperative to the usefulness of the program. A saved Origrammer file had to contain all relevant model information like the title, author, comments, recommended paper size, paper shape, paper color as well as the diagram information itself. Every line, arrow, symbol and the textual instructions for each step had to be included and correctly displayed after opening a file.

5.4 Crossing Lines

When inputting several overlapping lines, they should be split up and a new addressable vertex at their crossing points has to be created. Implementing this feature by itself was done relatively quickly. However, with the addition of the endpoint offsets for existing creases (see Section 2.1 *Creases should not contact the edges*) more variables had to be kept in mind. As the user can freely draw lines in different directions and in random order, a lot of edge cases had to be considered (see Fig. 5.2). In the Origrammer, a basic line is represented, amongst other things, by two points and two boolean values in case the lines should be drawn with an offset (for the previously mentioned existing creases).

```
OriLine(p0(x,y), p1(x,y), boolean startOffset, boolean endOffset);
```

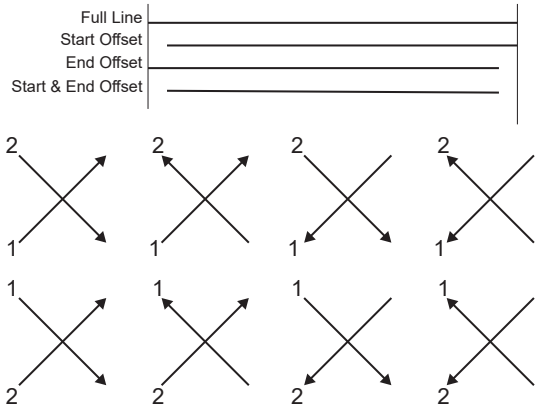


Figure 5.2: Crossing Lines Input Possibilities

So once two lines intersect each other, they have to be split up into four

shorter lines. In order to accomplish this their `crossingPoint` is calculated and the new lines are created (see Fig. 5.3). Important here is, that the endpoint that contains the crossingPoint can not have an offset. This is also the point where the previously shown variety of different user inputs has to be taken into consideration. The start- and endpoints of a line change depending on what direction the user inputs them. Consequently a test had to be implemented that checks which direction a line in question is facing and that adjusts following calculations accordingly. Otherwise the offsets would be either drawn in the wrong order or too many offsets in general would be drawn. Additional complexity arises once an input line crosses multiple existing lines. In this case the input line has to be split into multiple smaller lines - at each crossing point of another existing line.

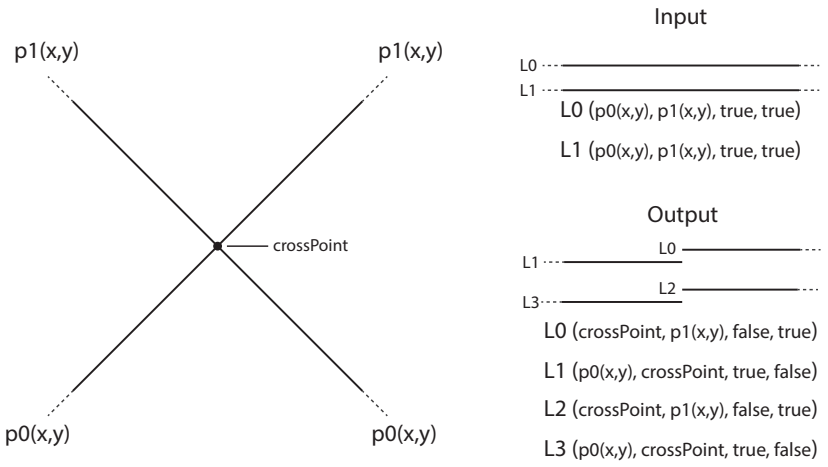


Figure 5.3: Crossing Lines Example

6 Prospect

Despite the mostly finished state of Origrammer there are various additional features that could prove useful once implemented. This section will list and explain missing features and general usability improvements and the following subsections are roughly sorted by descending importance. Though this should not be seen as an extensive list of missing essential features, but rather as a prospect on what the further development of this diagramming program could look like.

6.1 Export to .pdf

Exporting a finished diagram to the Portable Document Format (PDF) would provide an easy way to share diagrams created within the Origrammer. This would provide greater accessibility to the diagrams while also removing the requirement of using the Origrammer just for displaying purposes. This would be especially helpful for folders who simply want to follow the diagrams.

6.2 "Intelligent" Input Functions

So far every line, arrow or symbol has to be manually placed by hand. In order to increase productivity some "intelligent" functions could be implemented. These functions could for example place, rotate and scale arrows automatically after placing a valley/ mountain fold on the paper.

6.3 Input Validation

As already mentioned in Section 5.2, a proper input validation is currently missing in a couple of places.

6.4 Further Polishing

6.5 General Usability Improvements

An improvement of the usability of Origrammer can be achieved by a multitude of measures. Simple tooltips or short explanatory texts would remove some uncertainty from novices as well as giving reminders for experienced

users. When taking this further, a help section that explains all of Origrammers features in greater detail could be implemented. This section could also visually show how to use the features, which would remove the barrier of having to read long texts that can act as a deterrent especially for newer users.
Using current Human Computer Interaction (HCI) methods to increase usability would also be a logical step in the future. This process can be further improved by conducting external usability tests or creating surveys.

Glossary

Arrows of Action Arrows of Action indicate an action performed on the paper.. 9

Arrows of Motion Arrows of Motion indicate where the paper folded towards.. 9

Mountain Fold The fold that results after folding one part of the paper behind the other. After unfolding you roughly see an A-shape. 6, 8

Origami (jpn: *ori* = *folding* and *kami* = *paper*) is the art of folding paper into models of animals, people or other objects.. 4

Origrammer Origrammer is the name of the diagramming program, developed within the scope of this project. 17, 25

Rabbit Ear A Rabbit Ear is a folding sequence, that narrows the paper and creates a new flap. This is done by creating three folds at the bisectors of a triangle.. 13

Valley Fold The fold that results after folding one part of the paper over the other. After unfolding you roughly see a V-shape. 8

X-Ray Line An X-Ray line indicates a fold or edge that is hidden behind a layer of paper. 8

List of Figures

List of Tables

References

[1] Robert J. Lang. Origami diagramming conventions, 2011. http://www.langorigami.com/article/origami-diagramming-conventions.

[2] Akira Yoshizawa. *Atarashi origami geijutsu.* Origami Geijutsu-Sha, Tokyo, 1954. no ISBN.