Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS

Martin Pecka

Origami diagram creator

Department of Software and Computer Science Education

Supervisor of the bachelor thesis:	Mgr. Martin Petříček
Study programme:	Informatika
Specialization:	Obecná informatika

Prague 2011

Poděkování.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague date ....

Název práce: Origami diagram creator

Autor: Martin Pecka

Katedra: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Martin Petříček, Kabinet software a výuky informatiky

Abstrakt: Program Origamist si klade za cíl pomoci s tvorbou návodů na skládání origami modelů. V současné době jsou nejběžnějšími metodami pro tvorbu těchto návodů buď ruční kreslení všech kroků v obrázkovém editoru, nebo nafocení jednotlivých kroků a jejich manuální poskládání (opět v obrázkovém editoru). Origamist na toto pole přináší novou alternativu. Autor tak dostává možnost přenést posloupnost ohybů papíru, z nichž se návod skládá, do programu Origamist, jenž z nich dokáže vygenerovat několik druhů výstupu - v origamistických kruzích nejrozšířenější PDF návod, ale i návod jako obrázek (PNG, SVG), nebo dokonce jako animaci procesu skládání. Přidanou hodnotou pak je snadná možnost přeložit popisky kroků do více jazyků.

Klíčová slova: Origami, Java3D, skládání papíru, návod

Title: Origami diagram creator

Author: Martin Pecka

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Martin Petříček, Department of Software and Computer Science Education

Abstract: The main target of the Origamist application is to aid with creating origami diagram manuals. Recently, the most common methods for creting those manuals are drawing of the steps in an image editor, or taking photographs of the folded steps and composing them together (again in an image editor). What Origamist brings is a new alternative to these methods. The manual's author gets the possibility to transfer the sequence of paper folds the manual consists of to the Origamist application, which is able to export several types of output - the most favourite (among origami folders) PDF manuals, but also image manuals (as PNG or SVG), and even as an animation of the folding process. There is also the added value to translate the steps' descriptions to several languages.

Keywords: Origami, Java3D, paper folding, diagram

Contents

Introduction

Everyone who has ever tried to fold an origami model knows how important it is to have a *good manual* describing the steps of the folding process. A good manual consists of lots of images (generally one image per step) and their descriptions.

Why are the manuals needed? It is too hard (if not impossible) for most people to guess the folding process by only seeing the result shape. And this is even harder if you only see the result as a 2-dimensional image on screen or paper. Thus, several methods to aid people with the folding process were introduced.

The most practical help method is learning from someone who knows how to fold the desired model. This method has two disadvantages - a man can forget what he has learnt, and, in most cases, there is simply nobody who knows how to fold the desired model (except in origami communities). Therefore *paper manuals* were invented. They are (conceptually) eternal and are relatively easy to obtain (in books or on the Web). They have another disadvantage - if some steps are unclear in the manual (which happens not so infrequently), there is no other help. The last (relatively new) means to learn the folding process are *video tutorials*.

In the paper manuals (or bitmap image manuals, we will call both of them 'paper' manuals) there are some established graphical marks that indicate the operations to be done with the paper in the step. We will discuss these marks in higher detail further in the text. The marks cover the most of operations one would like to do with the paper, but their meanings aren't fixed and unambiguous, which can lead to lack of clarity of the manual.

'Paper' and video manuals share one more disadvantage, too. They aren't *simply editable* by other people. The 'paper' manuals are either printed or distributed as PDF or image files, and these aren't simply editable (PDF editors exist, but aren't widely used; editing an image manual involves some non-basic knowledge of computer usage). Video editing is even more difficult. So, ways to edit these manuals exists, but none of them is straightforward.

Origamist brings a new alternative to those types of manuals. It presents the concept of 'live' manual. Each folding step is represented as a 3-dimensional model, which the user can view from different viewing angles and zoom levels. Furthermore, everyone can simply edit the model in the Origamist editor. It doesn't matter if the user just wants to add a translation of the steps' descriptions, edit existing step descriptions or if he wants to add some more steps, all of these activities can be done straightforward in the editor.

Also, all of the previously mentioned types of origami manuals (except personal assistance) can be exported from the Origamist application. Only the exported animation has no sound track, which is an important part of video tutorials (but it is possible to add this functionality, also the data model can be simply modified to store this type of descriptions).

1. What is origami?

Origami is the name of an ancient Asian art of folding various figures and shapes from a piece of paper.

"Whether it is called ‘zhe zhi,’ as it is by Mandarin-speaking Chinese, or ‘chip chee,’ as Chinese who use the Cantonese dialect call it, or by the Japanese name ‘origami,’ it is generally agreed that the art of paperfolding originated in China perhaps before the 6th century." [1, p. 123]

The traditional Japan origami focuses mainly on animal figures and flowers. You could know the most famous origami animal - the crane. It is considered as a sample of the traditional technique.

In recent times, hundreds of other origami models can be found, including boxes, decorations and ornaments, envelopes, abstract things and much more. [2]

1.1 The rules of traditional origami

There aren't much constraints in the traditional origami. Everyone can fold whatever his fantasy invents, but to call the model a traditional origami, there are these rules: [3]

- To begin with a single square sheet of paper.
- Not to tear or cut the sheet of paper.
- Not to use glue.

And that's it. No more constraints on what can be done. This is also the set of rules Origamist tries to hold (in fact, it allows non-square papers).

1.2 Other kinds of origami

Besides the traditional origami, there are several other types, differing in what is allowed or disallowed. Here is a short list of some other techniques:

- *Modular origami*, which allows to use and combine multiple sheets of paper. [5]
- *Action origami*, which creates figures whose parts are moveable, so they can be used as toys. [6]
- *Pureland origami*, which only allows one fold to be done at once, and thus disallows folds like reverse folds or rabbit folds. This type of origami is suitable for beginners, children and disabled people. [7]
- *Kirigami*, which allows cutting the paper and using glue. Kirigami is mostly used to make decorations. [8].
- *Technical origami*, which develops the models based on computer-generated crease patterns[1]. [9].

[1] More on crease patterns can be found further in section 1.5

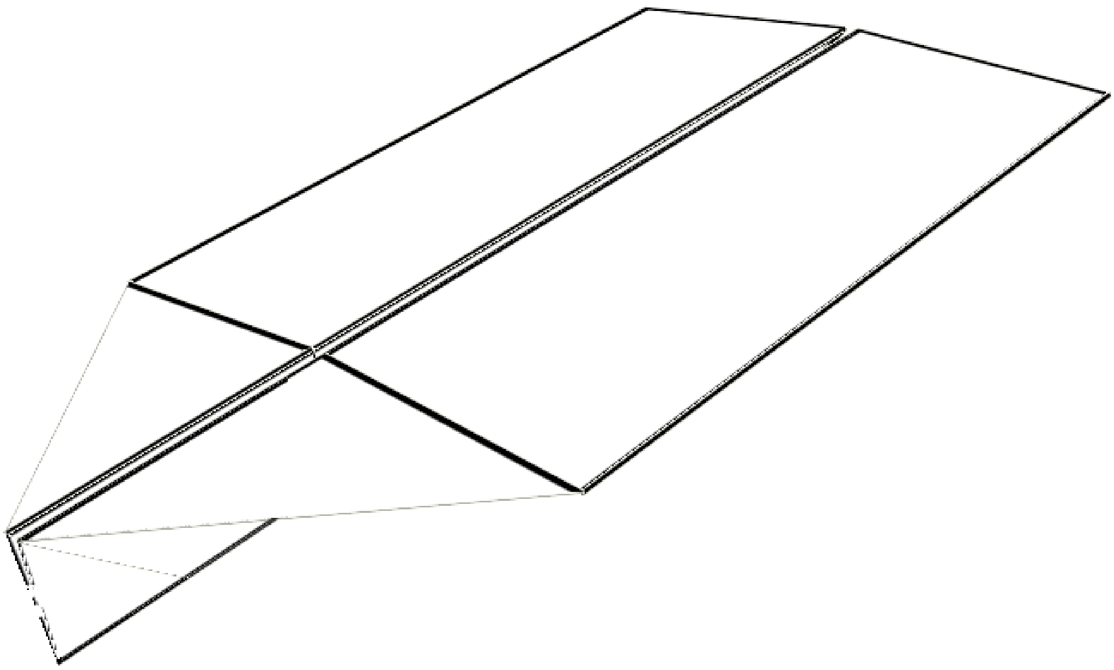
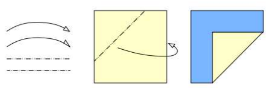


Figure 1.1: Origami model of paper plane

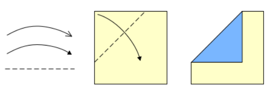
- *Mathematical origami*, which is just abstract and is used for some mathematical and algorithmic proofs.
- *Rigid origami*, which is a subset of mathematical origami and tries to find answers to the question "If we replaced paper with sheet metal and had hinges in place of the crease lines, could we still fold the model?" [9].

1.3 Basic folds

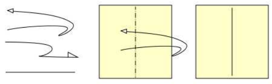
In this section, some basic fold types will be described, along with the common marks for them. Sample images are taken from [10], the steps descriptions are inspired by [2].



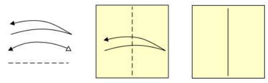
Mountain fold. This is a simple fold which bends the paper in the direction ‘from the viewpoint’. An arbitrary angle can be specified for this type of fold.



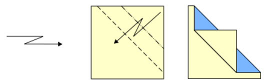
Valley fold. This is a simple fold which bends the paper in the direction ‘towards the viewpoint’. An arbitrary angle can be specified for this type of fold.



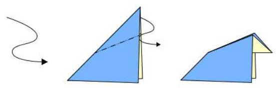
Mountain fold and unfold. This fold just makes a mountain crease on the paper. After doing it, the paper has the same geometry as before, but the crease has required moving the paper. This is mainly used for creating reference creases.



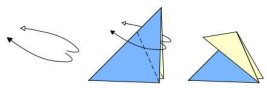
Valley fold and unfold. This fold just makes a valley crease on the paper. After doing it, the paper has the same geometry as before, but the crease has required moving the paper. This is mainly used for creating reference creases.



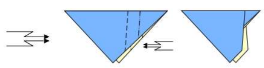
Thunderbolt fold. A double fold, one of the folds is mountain, the other is valley An arbitrary angle can be specified for both folds (a different one for each of them).



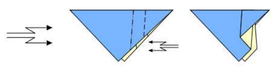
Inside reverse fold. Tuck a tip of a flap[2] inside. Uses two mountain folds for side creases, and a valley fold for the inner centre line. For every paper configuration, there exists only one angle for the created folds which guarantees to preserve the paper properties.



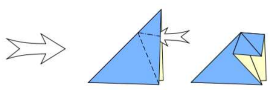
Outside reverse fold. Wrap a tip of a flap[2] around its outer side. Uses two valley folds for side creases, and a mountain fold for the outer centre line. For every paper configuration, there exists only one angle for the created folds which guarantees to preserve the paper properties.



Inside crimp fold. This is a double fold where both folds are inside reverse folds.

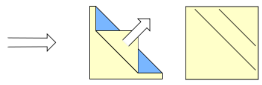


Outside crimp fold. This is a double fold where both folds are outside reverse folds.

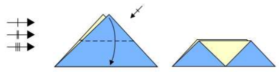


Open fold. Open or squash a flap[2] of paper. Origamist doesn't support this operation.

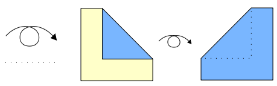
[2] A triangle standing out of the paper, which consists of at least two paper layers.



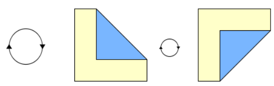
Pull fold. Unfold some previously created folds. Origamist has only partial support for this operation. Only one fold can be unfolded at a time, so eg. atomically unfolding a thunderbolt fold isn't possible.



Repeat. Repeat some previously done operations. This operation hides (for clearness) some steps and substitutes them with the repetition mark.



Turn over. Turn the paper to the other side. Origamist only supports turning around the horizontal axis of the current view, and a single turning angle - 180 degrees.



Rotate. Rotate the model of an arbitrary angle around the current view's direction axis (normal of the screen).

By using these operations, it is possible to create most of the basic and moderate origami models. The advanced models, however, often require the use of more advanced techniques. Some of them are described in [2].

1.4 More on operations' marks

As can be seen from the above images (if they aren't too small), some folds are drawn as solid lines, other are drawn as dashed lines and so on. Here are the rules for deciding how to visualise a fold line. The edges of paper are always drawn solid, a little thicker than other lines. Visible creases made in the previous steps (except the current one) are drawn by solid lines (or they can be completely omitted). Invisible[3] folds are drawn by a dotted line (.....). And finally, the creases made in the current step of the manual are drawn as dashed lines (—–) for valley folds and dash-dot lines for mountain folds (-..-..-). Note that the mountain/valley assignment is relative to the point of view.

1.5 Crease patterns

Crease patterns, although they are a very old concept, experience a boom in the latest years as an alternative to classical origami manuals. What is a crease pattern? It is the unfolded origami model, which retains the creases created during the folding process. So, basically, a crease pattern is a set of lines on a

[3]Folds hidden under other parts of paper. The visualisation using dotted lines is sometimes called X-Ray folds.

straight sheet of paper. Depending on the purpose, the directions of the folds (whether they are mountain or valley) may or may not be specified.

Why are they so popular? This may be due to two wholly different reasons. The first one is that folding a model (with known target shape) from a crease pattern is a challenge, and has no direct instructions. The second reason is that crease patterns are very important in mathematics and computational origami, which is shown in the next chapter.

2. Origami, computers and mathematics

This chapter presents some recent origami-related results in the field of mathematics or computer science, and also presents some existing origami software. Although origami is mainly art, most of these results are very interesting, and some of them have even practical use.

2.1 Computational origami

"Most results in computational origami fit into at least one of three categories: universality results, efficient decision algorithms, and computational intractability results.

A *universality result* shows that, subject to a certain model of folding, everything is possible. For example, any tree-shaped origami base, any polygonal silhouette, and any polyhedral surface can be folded out of a large-enough piece of paper. Universality results often come with efficient algorithms for finding the foldings; pure existence results are rare.

When universality results are impossible (some objects cannot be folded), the next-best result is an *efficient decision algorithm* to determine whether a given object is foldable. Here 'efficient' normally means 'polynomial time'. For example, there is a polynomial-time algorithm to decide whether a 'map' (grid of creases marked mountain and valley) can be folded by a sequence of simple folds.

Not all paper-folding problems have efficient algorithms, and this can be proved by a *computational intractability result*. For example, it is NP-hard to tell whether a given crease pattern folds into any flat origami, even when folds are restricted to simple folds. These results mean that there are no polynomial-time algorithms for these problems, unless some of the hardest computational problems can also be solved in polynomial time, which is generally deemed unlikely." [11]

2.1.1 Other computational tasks

As [11] says, there is another division of computational problems. They are *origami design* and *origami foldability* problems. *Origami design* examines the possibilities (finding the needed creases) of folding arbitrary shapes with specified properties. On the other side, *origami foldability* tries to answer if the given crease pattern is foldable into any shape.

One complete straight cut problem solves the task whose input is a polygon drawn on paper, and whose output is a crease pattern which can be folded in the way that all the polygon's edges overlap on a single line. Why one straight cut? If you cut the paper along the overlap line, you will get the polygon cut out of the paper. [11] presents results of two different approaches, both of them proving that such crease pattern can be found for any polygon.

Silhouettes and polyhedra problem takes a 2- or 3-dimensional silhouette of the desired model as its input and returns a folding of the paper resulting in the input shape. Again, according to [11], such result always exists.

Flat foldability tries to find out if a given crease pattern (without fold directions specified) is foldable to a flat model using a consistent fold direction assignment. [12] shows a linear-time algorithm that finds such direction assignment, or tells that the given pattern is not flat foldable.

2.1.2 Existing origami software

There is a number of computer applications related to origami. Here is a list of some of them.

- *TreeMaker*[1] by Robert Lang, is a tool for designing crease patterns for given sketched shapes. The shape is basically sketched as a tree (in the algorithmic meaning of the word), thus TreeMaker. The program uses the disc packing method, described in [13].

- *Origami simulation*[2] by Robert Lang is a simple GUI tool for designing pureland origami. Since 1992 the development has been stopped, and it only works on PowerPC Macs.

- *ORIGAMI Playing Simulator in the Virtual Space* by Shin-ya Miyazaki, Takami Yasuda, Shigeki Yokoi and Jun-ichiro Toriwaki. Another interactive GUI tool for origami model folding, was presented in [14]. Although this seems to be the most capable GUI tool for origami, I couldn't test it, because the given paper doesn't provide a link to the application.

- *Doodle*[3] by Jérôme Gout, Xavier Fouchet, Vincent Osele, and volunteers. Doodle is a tool for generating origami diagrams from the given ASCII text instructions. This program produces nice diagrams, but needs a lot of additional information in the input files (eg. it is often needed to rotate some paper parts by an explicit command, because the fold commands just create the associated fold lines).

- *ORIPA*[4] by Jun Mitani. ORIPA is interactive GUI tool for designing crease patterns. It also provides an estimated preview of the folded model.

- *Computational Origami System Eos* by Tetsuo Ida, Hidekazu Takahashi, Mircea Marin, Asem Kasem and Fadoua Ghourabi. This software introduced in [15] is a non-interactive tool for folding origami models and automatic proving some conclusions about the constructed models. EOS is written in Mathematica.

2.2 Origami in other sciences

Mathematics and computer science aren't the only disciplines origami brings benefits to. Here is a list of other uses of origami theory:

[1] `http://www.langorigami.com/science/treemaker/treemaker5.php4`

[2] `http://www.langorigami.com/science/origamisim/origamisim.php4`

[3] `http://doodle.sourceforge.net/about.html`

[4] `http://mitani.cs.tsukuba.ac.jp/pukiwiki-oripa/index.php?ORIPA%3B%20Origami%20Pattern%20Editor`

- *Stents* in medicine. Stents are small tubes used to reinforce or broaden clogged veins and arteries. Since they are transported to their destination through veins and arteries, they need to be in a folded state, then they can be transferred to the right place, and unfolded. A waterbomb base[5] is mainly used for stents. [16]

- *Eyeglass space telescope* needed the help of origami theory master Robert Lang[6] to invent a method of folding a large telescope into a space satellite. [17]

- *Space project SFU* of Japanese space organization JAXA used Robert Lang's origami experiences to efficiently fold and unfold a solar panel array in the open space. The fold they used is called Miura map fold[7].[18]

- Origami design techniques were used by a German company to simulate effective *packing of airbags* into car steering wheels. [19]

- Origami techniques helped with the development of *3D solar panels*, which increase the productivity of solar panels and have the advantage of not having any movable parts that can get broken. [20].

[5]One of the standard origami bases, which are just a series of steps shared by many models in the beginning.

[6]`http://www.langorigami.com/science/eyeglass/eyeglass.php4`

[7]`http://en.wikipedia.org/wiki/Miura_map_fold`

3. Origamist basic facts

Here I will present the basic structure of Origamist and the technologies used.

3.1 Basic structure of Origamist

3.1.1 Two parts of the application

Origamist isn't just a single application. It consists of two standalone parts.

Origamist editor provides the tools needed to create new origami diagrams and manuals. So various folds can be added to existing models, or a brand new model can be created. Also the metadata of the model can be edited.

Origamist viewer is a viewer application, that can only display formerly created diagrams. Its main target is to be used as webpage applet, so that the visitors of an origami site can comfortably browse the models the site offers. Although the viewer isn't primarily meant to create the manuals, it has the same export possibilities as the editor, so every user of any part of Origamist is able to create the manuals in various export formats.

3.1.2 Programming language and deployment technologies

Origamist has been written mainly in Java[1]. There were several reasons for this decision. Firstly, Java is a multiplatform language ([21]), which is a basic requirement due to the intended viewer usage. Secondly, Java applications can be integrated directly into a webpage (using the Java Web Start or Java Plugin), which rapidly simplifies the access to the application. The third most important reason was that no platform-specific code has to be written (at least not in Origamist[2]), which greatly simplifies the programming work and allows to concentrate on the core business.

Java provides multiple solutions to application deployment, and Origamist uses these three (or four) of them:

- The first one is a standard Java application packed into a single JAR archive file[3]. So, anyone can run the application by moving to the folder containing the JAR and typing:

  ```
  java -jar OrigamiEditor.jar
  ```

- Another possibility is to include the application into a webpage as Java applet using the standard applet code. See 4.2 for a sample webpage.

- A similar one is to include the application into a webpage as Java applet using the *Next-Generation Java Plugin Technology* introduced into Java in version 6u10 ([22]). See 4.3 for a sample webpage.

[1] Several other languages have been used, such as XSLT and XML, but these are only scripting and tagging languages not providing the core functionality.

[2] Otherwise, Java allows to use platform-specific code, but doing so isn't a common practice in Java.

[3] Application libraries are standalone files, too.

- The last possibility is to launch and install the application through the *JNLP protocol*[4]. This protocol allows Java applications to be downloaded and installed on the target machine as standalone applications, so the end user can eg. make shortcuts to these programs and utilise them offline. JNLP also provides a way of automatic updates of the downloaded software. See 4.4 for a sample JNLP file that launches Origamist viewer.

3.1.3 Data files

Origamist saves all its data[5] in XML text files. There are two types of files Origamist recognises.

Diagram files

Diagrams are XML files with the *.xml* file extension having their root element in a model-specific namespace. For technical details on this, consult the programmer's manual.

The XML file contains all information needed to render the model and export the manuals, and also some metadata[6].

These files follow the convention of not storing any 3-dimensional data, so all folds are defined only by their position on the crease pattern[7].

Listing files

Origamist recognises one special type of XML files - those having their name *listing.xml* and having their root element in the listing-specific namespace. For technical details on this, consult the programmer's manual.

Listing files are used by the viewer to define and organise whole sets of models. The listing files have a tree structure, where every model can be attached to a named category. Some excerpts of the models' metadata are also saved in the listing files, so that it isn't required to download the whole model to see its name or thumbnail.

Origamist provides no complete support for creating these files. Here are two main reasons. These files are supposed to be automatically generated on a server providing the models (this is the expected use case). Moreover, a partial support for manual creation of these files exists. If the user loads a whole directory structure into the viewer, it automatically creates categories for matching subdirectories using their names. The listing can then be saved from Origamist viewer, however it doesn't support extended features like fully localised category names[8].

---

[4] `http://www.oracle.com/technetwork/java/javase/index-142562.html`

[5] Except user preferences which are stored in the system registry.

[6] Like the model's name, author's name, description of the model, paper formats, licensing information and so on.

[7] More on this topic in the programmer's manual, section ??

[8] These can be added manually to the exported listing file.

3.2 Interesting technologies used in the project

The most interesting technologies Origamist uses are listed here, for a full list of technologies and libraries, refer to the programmer's manual.

3.2.1 Git

Git is a distributed versioning system which I have used for recording the work progress. It is easy to use and does its work well. Sometimes, the ability to amend last commit[9] is useful and helps keeping the repository clean. Even though nobody else used the repository and I have been the only commiter, the repository gave me bigger freedom in what I do with the code. The whole Git repository is on the attached CD.

3.2.2 JAXB

Java And XML Bindings[10] is a library for mapping XML files to Java objects and vice versa. It not only provides the mapping, but, since the XML files' schemata and the Java classes carry redundant information, offers to generate either the schemata or the Java classes.

I have chosen to write schemata for the data files (they are in XSD[11] format), and JAXB[12] generates the Java classes automatically. The generated classes are plain Java objects annotated by some special annotations, and they follow the Bean pattern[13]. Advantages of this approach are significant if there is the need to add some fields to the data files[14], just edit the XML schemata and the Java classes will be updated to be consistent with the schemata without any additional effort.

Among other advantages I would like to stress the simplicity of the Java $\leftrightarrow$ XML conversion[15]. It basically consists of just a few lines of code[16], but allows a great deal of customisation if it is needed.

Nevertheless, JAXB has one disadvantage. Since the classes are auto-generated, there is no chance of adding further functionality directly to those files, so they always remain plain 'data envelopes'. Although this may seem to be a big disadvantage, there is a near elegant solution to this problem. See ?? for detailed information.

[9] Additionally change the files or comment of the last commit.

[10] `http://www.oracle.com/technetwork/articles/javase/index-140168.html`

[11] XML Schema Definition, which is a promising successor of the old DTDs

[12] Using its tool XJC

[13] `http://download.oracle.com/javase/tutorial/javabeans/whatis/index.html`

[14] Which is often during the initial development, but no so frequent in further 'life' of the application

[15] JAXB has the terms marshalling for Java $\rightarrow$ XML, and unmarshalling for XML $\rightarrow$ Java

[16] Refer to ~services.JAXBListingHandler and ~services.JAXBOrigamiHandler

3.2.3 Java3D

Java3D[17] is a powerful Java library for displaying and creating 3D graphics. It provides two basic modes of work — a 'direct' mode[18] which makes working with Java3D much like working in OpenGL; the other mode is more object-oriented[19] and provides an easy-to-use interface for composing the 3D scene, which means that most of the computations are hidden before the programmer. Java3D integrates nicely with the Swing[20] GUI [21] library.

Since 3D graphics is basically a very low-level work, Java3D needs some native libraries to run. This effectively narrows the list of platforms Origamist will run on to the list of Java3D supported platforms. Fortunately, a Java3D implementation is available for most of the Java supported platforms [23]. As native libraries cannot be distributed in the standard way as JAR libraries are, they require special handling. Refer to the programmer's manual for more on this.

3.2.4 Forms

JGoodies Forms[22] is a Swing layout manager. Swing provides some layout managers in the standard JRE[23] distribution, but they either aren't much flexible, or are overly complex. JGoodies Forms allows the programmer to define a grid[24] on the current window or component, and put other components into this grid.

3.2.5 Batik

Batik[25] provides a simple way of generating SVG[26] and PDF files in Java. It has other SVG-related capabilities, too, but they are unimportant for this project.

3.2.6 Ant

Apache Ant[27] is a Java build tool. Its design is very robust, it allows to trigger innumerable different tasks during the build, so that everything needed to build and deploy a complex Java application is to run command

```
ant
```

in the command line from the application's base directory. Ant even supports plugins[28].

[17] `http://java3d.java.net/`
[18] Called 'immediate mode' in Java3D.
[19] Called 'retained mode' in Java3D.
[20] `http://download.oracle.com/javase/tutorial/uiswing/start/about.html`
[21] Graphical User Interface
[22] http://www.jgoodies.com/freeware/forms/
[23] Java Runtime Environment, the minimal installation of Java needed to run most Java applications.
[24] The use of grids is very common in UI design, but JRE doesn't provide any layout manager capable of working with more complex grids.
[25] `http://xmlgraphics.apache.org/batik/`
[26] Scalable Vector Graphics
[27] http://ant.apache.org/
[28] And this project uses some of them.

4. Origamist's algorithms & data structures

This chapter covers the most important computational algorithms, data structures and programming approaches the program uses.

4.1 Representation of the origami model

4.1.1 Triangles and layers

The model is stored simultaneously as a 3D model and 2D crease pattern. Both these models are represented by a set of triangles. Every 2D triangle corresponds to exactly one 3D triangle, and thus we will call the 2D triangle as the 'origin' or 'original' of the 3D triangle (because in the very first step, the corresponding 2D and 3D triangles are the same). The triangles are grouped to so called *layers* of paper.

Definition. *A* layer of paper *is a nonempty set of triangles meeting the following conditions:*

- *All triangles have their normals pointing in the same direction*[1]*.*
- *The original triangles form a single, nondegenerated and connected polygon*[2] *on the crease pattern.*
- *The 3D triangles form a single, nondegenerated, connected and planar polygon.*
- *Every triangle belongs to exactly one layer.*

As a consequence of this definition, we see that the whole paper model can be parcelled into layers. A layer of paper can be imagined as the largest straight part of the paper bounded by creases or edges of the paper[3].

What is the term of layer good for? A layer is always the smallest unit of paper that can be moved, bent, or rotated. If a fold would go through the interior of a layer, a crease is created in the layer and it is subdivided into more smaller layers[4].

4.1.2 Fold lines

Although it is not necessary to hold all the fold lines in memory[5], it shows that it is helpful to have a quick access to them. So, every triangle remembers all the

[1]This trivially holds for the original triangles

[2]But possibly non-convex.

[3]But it is allowed for a layer to have its boundary at a crease of angle 0°. This means it doesn't always have to be the largest straight part, but it is guaranteed that a layer doesn't have its boundary somewhere 'inside' (the boundary is always an edge of the paper or a crease).

[4]Convex layers always split to 2 parts, but nonconvex layers can generate more sublayers

[5]All fold lines could be just signalised by layers' boundaries.

fold lines it lies along, and all fold lines have links to the triangles lying along them. Moreover, the folds remember the direction of the fold (mountain/valley), so it can be used later in generating crease patterns or for displaying the just done operations (as described in 1.4). They also remember their 'age' in the number of steps they were last 'touched' (used in a fold). The age can be used for blending old and probably unimportant creases.

4.1.3 That's all we need

The triangles, layers and fold lines are all we need to know to be able to correctly render and alter the model. More precisely, only 3D triangles and fold lines are needed, but layers and 2D triangles are needed for interaction with the model.

Závěr

Bibliography

[1] TEMKO, Florence; JACKSON, Paul. *Paper pandas and jumping frogs.* China : China books & periodicals, 1986. 133 p. ISBN 0-8351-17701770-7.

[2] LANG, Robert J. *Origami design secrets : mathematical methods for an ancient art.* Massachusetts (USA) : A K Peters, Ltd., 2003. 585 p. ISBN 1-56881-194-2

[3] PEN, K. *Origami Paper Folding: Rules of the Game.* Associated content [online]. 2009 [cited 2011-05-25]. Available at WWW: `http://www.associatedcontent.com/article/1818997/origami_paper_folding_rules_of_the.html?cat=24`.

[4] ORACLE *Supported system configurations for Java SE 6 and Java For Business 6* [online] [cited 2001-05-25]. Available at WWW: `http://www.oracle.com/technetwork/java/javase/system-configurations-135212.html`

[5] FUSE, Tomoko. *Unit Origami: Multidimensional Transformations.* Japan : Japan Publications, 1990. ISBN 0870408526.

[6] LANG, Robert J. *Origami in Action: Paper Toys that Fly, Flap, Gobble and Inflate.* Massachusetts (USA) : St. Martin's Griifin, 1997. ISBN 0-312-15618-9.

[7] SMITH, John S. *Pureland Origami.* Great Britain: British Origami Society, 1980.

[8] TEMKO, Florence. *Kirigami Home Decorations.* China : Tuttle Publishing, 2006. ISBN 0-8048-3793-7.

[9] MOBILEREFERENCE. *Asian Art.* USA : SoundTells, 2003.

[10] WIKIPEDIA CONTRIBUTORS. *Origami techniques.* Wikipedia, The Free Encyclopedia [online]. 2011 [cited 2011-05-25]. Available at WWW: `http://en.wikipedia.org/w/index.php?title=Origami_techniques&oldid=420452758`

[11] DEMAINE, E. D.; DEMAINE, M. L. *Recent results in computational origami.* In Origami : Proceedings of the 3rd International Meeting of Origami Science, Math, and Education. California (USA) : Monterey, 2001. pp. 3–16. Available at WWW: `https://encrypted.google.com/url?sa=t&source=web&cd=1&ved=0CBsQFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.100.8809%26rep%3Drep1%26type%3Dpdf&ei=bLHcTfeRHM3JsgaLt4TlDg&usg=AFQjCNERXS_4tZRQ8ANq72-gMpIpBor1Jg`.

[12] BERN, Marshall; HAYES, Barry. *The complexity of flat origami.* In Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 175ďż˝183, Atlanta, January 1996. Reprinted in Proceedings of the 3rd International Meeting of Origami Science, Math, and Education, 2001

[13] Lang, Robert J. *TreeMaker 4.0: A Program for Origami Design.* [online] 1998 [cited 2011-05-25]. Available at WWW: http://www.langorigami.com/science/treemaker/TreeMkr40.pdf

[14] Miyazaki, Shin-ya; Yasuda, Takami; Yokoi, Shigeki; Toriwaki, Jun-ichiro. *An ORIGAMI Playing Simulator in the Virtual Space.* In The Journal of Visualization and Computer Animation, vol.7(1) Japan, 1996. pp. 25–42. Available at WWW: http://www.om.sist.chukyo-u.ac.jp/main/research/origami/journal/jvca.html

[15] Ida, Tetsuo; Takahashi, Hidekazu; Marin, Mircea; Kasem, Asem; Ghourabi, Fadoua. *Computational Origami System Eos* Japan : Department of Computer Science, University of Tsukuba. Available at WWW: http://www2.score.cs.tsukuba.ac.jp/publications/Eos.pdf/view

[16] You, Zhong; Kuribayashi, Kaori *A NOVEL ORIGAMI STENT* In 2003 Summer Bioengineering Conferencee, June 25–29. Florida (USA), 2003. pp. 257–258. Available at WWW: http://www.tulane.edu/~sbc2003/pdfdocs/0257.PDF

[17] Lawrence Livermore National Laboratory. *Eyeglass space telescope.* National Ignition Facility & Photon Science : Bringing Start power to Earth [online]. 2002 [cited 2011-05-25]. Available at WWW: https://lasers.llnl.gov/programs/psa/pdfs/technologies/eyeglass_space_telescope.pdf

[18] Shibayama, Y.; Arai, H.; Matsui, K.; Hama, K.; Ushirokawa, A.; Natori, M.; Takahashi, K.; Wakasugi, N.; Anzai, T. *SFU solar array.* European Space Power Conference. ESA Special Publication. 1989. , pp. 557-562. Available at WWW: http://adsabs.harvard.edu/abs/1989ESASP.294..557S

[19] GĎŻ˝rtner, Torsten; Eriksson, Magnus; FĎŻ˝ltstrĎŻ˝m, Jonas. EASi GmbH, *Advanced Technologies for the Simulation of Folded Airbags*, 2nd European LS-DYNA Conference, Gothenburg, Sweden, June 1999.

[20] Choi, Charles Q. *Solar Panel Productivity Boosted by Origami.* LiveScience.com [online]. 2010 [cited 2011-05-25]. Available at WWW: http://www.livescience.com/10625-solar-panel-productivity-boosted-origami.html

[21] Oracle *Supported System Configurations for Java SE 6 and Java For Business 6* Oracle.com [online]. [cited 2011-05-25]. Available at WWW: http://www.oracle.com/technetwork/java/javase/system-configurations-135212.html

[22] Oracle *Next Generation in Applet Java Plug-in Technology* Oracle : Sun Developer Network [online]. [cited 2011-05-25]. Available at WWW: http://java.sun.com/developer/technicalArticles/javase/newapplets/

[23] *Java 3D TM Downloads: Release Builds* java.net : The Source for Java Technology Collaboration [online]. [cited 2011-05-25]. Available at WWW: `http://java3d.java.net/binary-builds.html`

Attachments

4.2 A sample webpage displaying Origamist as a standard Java applet using the old Java Plugin

```
<html>
    <body>
        <applet code="cz.cuni.mff.peckam.java.origamist.gui.viewer.
           OrigamiViewerBootstrapper"
             width="800"
             height="600"
             archive="
               lib/log4j-1.2.16.jar,
               lib/j3dcore.jar,
               lib/j3dutils.jar,
               lib/vecmath.jar,
               lib/jaxb2-basics-runtime-0.6.0.jar,
               lib/forms.jar,
               lib/batik-anim.jar,
               lib/batik-awt-util.jar,
               lib/batik-bridge.jar,
               lib/batik-codec.jar,
               lib/batik-css.jar,
               lib/batik-dom.jar,
               lib/batik-ext.jar,
               lib/batik-gvt.jar,
               lib/batik-parser.jar,
               lib/batik-script.jar,
               lib/batik-svg-dom.jar,
               lib/batik-svggen.jar,
               lib/batik-transcoder.jar,
               lib/batik-util.jar,
               lib/batik-xml.jar,
               lib/itextpdf-5.1.0.jar,
               lib/JPEGMovieAnimation.jar,
               lib/pdf-transcoder.jar,
               lib/xml-apis-ext.jar,
               OrigamistViewer.jar
           ">

           <param name="files" value="diagrams/paper_plane.xml
              diagrams/advanced_diagram.xml diagrams/diagram.xml" /
              >
           <!-- These are the files to be displayed at startup. -->

```

```
            <param name="startupMode" value="page" />
            <!-- Startup mode of the page, either "page" or
            "diagram". -->

            <param name="java_arguments" value="-Xmx1024m" />
            <!-- Require more memory than the default portion. -->
        </applet>
      </body>
</html>
```

4.3 A sample webpage displaying Origamist as applet using new generation Java plugin, if the client has installed it

```
<html>
    <body>
        <applet code="cz.cuni.mff.peckam.java.origamist.gui.viewer.
            OrigamiViewerBootstrapper"
            width="800"
            height="600"
            archive="
              lib/log4j-1.2.16.jar,
              lib/j3dcore.jar,
              lib/j3dutils.jar,
              lib/vecmath.jar,
              lib/jaxb2-basics-runtime-0.6.0.jar,
              lib/forms.jar,
              lib/batik-anim.jar,
              lib/batik-awt-util.jar,
              lib/batik-bridge.jar,
              lib/batik-codec.jar,
              lib/batik-css.jar,
              lib/batik-dom.jar,
              lib/batik-ext.jar,
              lib/batik-gvt.jar,
              lib/batik-parser.jar,
              lib/batik-script.jar,
              lib/batik-svg-dom.jar,
              lib/batik-svggen.jar,
              lib/batik-transcoder.jar,
              lib/batik-util.jar,
              lib/batik-xml.jar,
              lib/itextpdf-5.1.0.jar,
              lib/JPEGMovieAnimation.jar,
              lib/pdf-transcoder.jar,
              lib/xml-apis-ext.jar,
              OrigamistViewer.jar
```

```
            ">

                     <param name="jnlp_href" value="origami_viewer.
                         jnlp">
                     <!-- This is the only difference between the old
                         and new generation plugins. Listing of the
                         archive attribute and filling up params in
                         this page isn't used by the new generation
                         plugin, but it is there to work either on user
                          clients with old plugins. -->

             <param name="files" value="diagrams/paper_plane.xml
                 diagrams/advanced_diagram.xml diagrams/diagram.xml" /
                 >
             <!-- These are the files to be displayed at startup. -->

             <param name="startupMode" value="page" />
             <!-- Startup mode of the page, either "page" or
             "diagram". -->

             <param name="java_arguments" value="-Xmx1024m" />
             <!-- Require more memory than the default portion. -->
         </applet>
      </body>
</html>
```

4.4 A sample JNLP file to run Origamist viewer

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jnlp PUBLIC "-//Sun Microsystems, Inc//DTD JNLP
    Discriptor 1.5//EN" "http://java.sun.com/dtd/JNLP-1.5.dtd">
<jnlp spec="1.0+" codebase="http://www.ms.mff.cuni.cz/~peckam/
    origamist" href="origami_viewer.test.jnlp"><!-- codebase must
    point to the !absolute! location of the JNLP file, otherwise
    the newest Java versions won't run it! -->
  <information>
    <title>Origamist viewer</title>
    <vendor>Martin Pecka</vendor>
    <homepage href="http://github.com/peci1/Origamist"/>
    <description>A viewer for digital origami models.</description>
    <description kind="short">An origami viewer application.</
        description>
    <offline-allowed/>
  </information>
  <security>
    <all-permissions/>
  </security>
  <resources os="Windows">
```

```
    <property name="sun.java2d.noddraw" value="true"/>
  </resources>
  <resources os="Mac OS X">
    <property name="j3d.rend" value="jogl"/>
  </resources>
  <resources>
    <j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se" initial-heap-size="128m" max-heap-size="1024m"/>
    <jar href="OrigamistViewer.jar" main="true"/>
    <jar href="lib/batik-anim.jar"/>
    <jar href="lib/batik-awt-util.jar"/>
    <jar href="lib/batik-bridge.jar"/>
    <jar href="lib/batik-codec.jar"/>
    <jar href="lib/batik-css.jar"/>
    <jar href="lib/batik-dom.jar"/>
    <jar href="lib/batik-ext.jar"/>
    <jar href="lib/batik-gvt.jar"/>
    <jar href="lib/batik-parser.jar"/>
    <jar href="lib/batik-script.jar"/>
    <jar href="lib/batik-svg-dom.jar"/>
    <jar href="lib/batik-svggen.jar"/>
    <jar href="lib/batik-transcoder.jar"/>
    <jar href="lib/batik-util.jar"/>
    <jar href="lib/batik-xml.jar"/>
    <jar href="lib/forms.jar"/>
    <jar href="lib/gluegen-rt.jar"/>
    <jar href="lib/itextpdf-5.1.0.jar"/>
    <jar href="lib/j3dcore.jar"/>
    <jar href="lib/j3dutils.jar"/>
    <jar href="lib/jaxb2-basics-runtime-0.6.0.jar"/>
    <jar href="lib/log4j-1.2.16.jar"/>
    <jar href="lib/pdf-transcoder.jar"/>
    <jar href="lib/vecmath.jar"/>
    <jar href="lib/xml-apis-ext.jar"/>
  </resources>
  <applet-desc name="Origami viewer" main-class="cz.cuni.mff.peckam.java.origamist.gui.viewer.OrigamiViewerBootstrapper" width="800" height="600" documentbase=".">
    <param name="files" value="diagrams/listing.xml"/><!-- These are the files to be displayed at startup. -->
    <param name="displayMode" value="PAGE"/><!-- Startup mode of the page, either "page" or "diagram". -->
  </applet-desc>
</jnlp>
```