E-Origami System *Eos*

Asem Kasem, Tetsuo Ida, Hidekazu Takahashi, Mircea Marin and Fadoua Ghourabi

We are developing a system called *Eos* (*E*-Origami System). *Eos* does what a human origamist would do with a piece of origami and moreover assists in reasoning about geometrical properties of origami construction. It is a collection of *Mathematica* programs specializing in computational origami. It has capabilities of symbolic and numeric constraints solving, automated theorem proving, visualization of origami constructions and web interfacing. In this paper we will describe the basic features of *Eos* in details.

1 Introduction

The discipline of studying and manipulating origami by computer systems has been termed as Computational Origami. We were interested in the usage of computers to simulate origami constructions that an orgamist will perform by hand, and moreover in reasoning about the geometric properties obtained by these constructions [5, 4]. We will present in this paper a system called *Eos*, E-origami system, which is developed using *Mathematica* and has several functionalities to manipulate and reason about origami computationally.

Eos has capabilities of symbolic and numeric constraints solving, automated reasoning and theorem proving, visualization of origami constructions and an interface to the web which offers users to have access to the functionalities of the system using a web browser.

The paper is organized as follows. In section 2, we will explain about our computational origami environment and its usage to construct origami pieces. In section 3, we will introduce our approach to use *Eos* as a theorem proving environment to prove the correctness of origami constructions. In section 4, we explain about webOrigami system that interfaces *Eos* to the web. And in section 5, we draw some conclusion and point out future research.

2 E-origami system *Eos*

Eos is developed as *Mathematica* packages specialized in origami processing which implements Huzita's six axioms [3], and Hatori's seventh axiom. Those seven axioms are considered the basic axioms to perform paper folding in our system. This feature of *Eos* is represented as a constraint solving problem to determine fold creases according to the constraints specifications that represent each axiom. These constraints are to be solved numerically to obtain fold creases during what we will be referring to as construction phase, and to be stored as geometric properties representing fold operations, and will be used later in reasoning about the construction, which we will be referring to as proving phase.

The axioms set is described as follows:

(O1) Given two points, we can make a fold along the fold line that passes through them.

(O2) Given two points, we can make a fold to bring one of the points onto the other.

(O3) Given two lines, we can make a fold to superpose the two lines.

(O4) Given a point P and a line m, we can make a fold along the fold line that is perpendicular to m and passes through P.

(O5) Given two points P and Q and a line m, we can make a fold to superpose P and m along the fold line that passes through Q.

(O6) Given two points P and Q and two lines m and n, we can make a fold to superpose P and m, and Q and n, simultaneously.

(O7) Given a point P and two lines m and n, we can make a fold to superpose P and m, such that the fold line is perpendicular to n.

Using this set of axioms, Origami constructions proceed stepwise, where each step indicates a fold operation that satisfies one of axioms (O1)–(O7). *Eos* provides a function `Fold` to realize folding axioms. In addition, every call of `Fold` has two effects: (1) visualizing the result of the fold step by computing the fold line and performing the fold, and (2) recording the geometric constraints that characterize the fold operation. In this way, *Eos* provides the user with interactive access to a view and manipulate the origami constructed so far, and to the collection of geometric constraints that describe this constructed origami.

Example of Constructing Regular Heptagon: We provide a stepwise execution of origami folds to demonstrate how to use *Eos* to construct regular heptagon using origami. The construction includes solving the angle trisection problem, by solving constraints represented as polynomials of degree 3. This is performed when applying axiom 6 as an intermediate step.
{*Full text of construction steps using Eos is attached in Appendix A*}

3 Theorem proving of the construction

After the origami construction phase using *Eos*, we can proceed to prove geometrical properties of the construction. A typical usage of *Eos* will proceed in the following steps:

Premise Generation: Extracting the geometrical properties of the construction and transform them into polynomial equalities and/or inequalities which form the premise of the theorem to be proved.

Conclusion Formulation: Representing the conclusion to be proved in polynomial equalities and/or inequalities.

Theorem Proving: Giving the premise and conclusion polynomials to a theorem prover to prove the conclusion. We distinguish the cases where the polynomials contain only equalities or also inequalities, as we will need to choose appropriate theorem prover for each case. For example, *Theorema* system provides Gröbner basis implementation to prove over equalities, and one might use *Mathematica*'s implementation of cylindrical algebraic decomposition when the theorem is partly described in inequalities.

Proof Generation: Depending on the selected theorem prover, the user obtains the proof result, and .

Note: Except for the conclusion formulation, all other proving steps are automatically performed.

4 *webOrigami*, Web Interface of *Eos*

In this section, we will present the current features of *webOrigami*, and we will spot light on the used technologies and design decisions.

Why *webOrigami* ?

We developed *webOrigami* project in order to enable origamists all over the world to have access to *Eos* functionalities without the need of having *Eos* packages or even any *Mathematica* installation requirements. By using a web browser (JavaScript Enabled), one can access the system and enjoy creating origami pieces by a simple interface that implements origami folding axioms, preview already constructed origami art pieces, and save construction steps into a *Mathematica* notebook that can be used later for reasoning purposes.

Figure. 1 shows a snapshot of the *webOrigami* page that you can access on the following URL:

`http://weborigami.score.cs.tsukuba.ac.jp`

What is *webOrigami* ?

webOrigami is our portal of computational origami project on the web. Basically, it is a client-server web application developed using the standard Java technology for web development, Java *Servlet*s and *Java Server Pages* (*JSP*), in corporation with *webMathematica* technology.

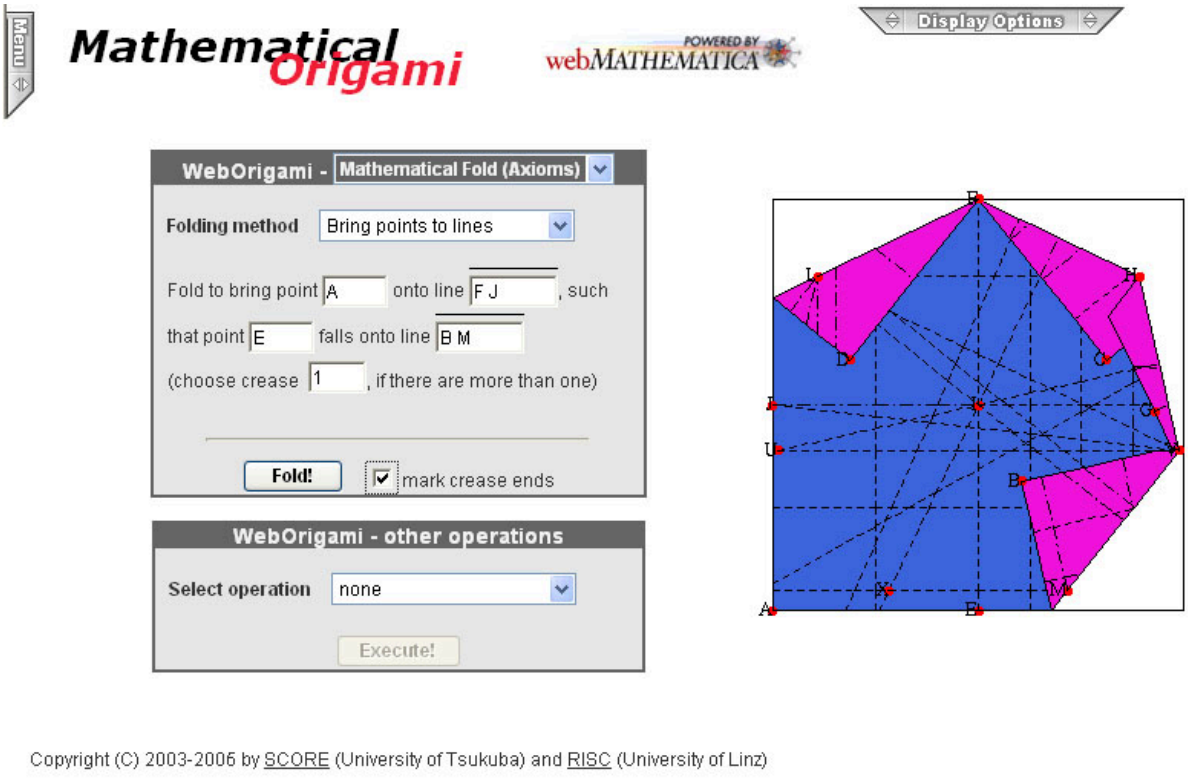


Figure 1: Web page of *webOrigami*

*Servlet*s are special Java programs that run in a Java-enabled web server, which is typically called a *Servlet* container. *JSP* technology provides a simplified fast way to create dynamic web content, by embedding Java code pieces in a normal *HTML* page. And *webMathematica*, which is based on Java's *JSP*s and *Servlet*s, allows a site to deliver *HTML* pages that are enhanced by the addition of *Mathematica* commands. Currently we are using *webOrigami* Apache Tomcat 5.5 as our web server and *Servlet* container.

An overview of how *webOrigami* site works is shown in Figure. 2.

1. A user sends request using a web browser to *webOrigami* located on a web server, requesting for specific origami operation.
2. *webOrigami* analyzes the request and acquires *Mathematica* kernel from a pool of available kernels through *webMathematica*.
3. The kernel then executes the requested operation using *Eos* packages and generates the proper output.
4. The web server returns the output to the user's web browser.

webOrigami Features and Design Decisions

webOrigami offers most features of *Eos* system to the web for constructing origami pieces. Those features include:

- *webOrigami* separates concurrent users by a username and password authentication.
- *webOrigami* enables users to create origami pieces with specific color choices for both faces of origami.
- Possibility to choose between classical origami folds and mathematical folds based on Huzita's axioms. Classical folds include mountain and valley folds which are specified by the fold line to fold through. Valley fold brings the faces separated by the fold line to face each other from the top view of the user, while mountain

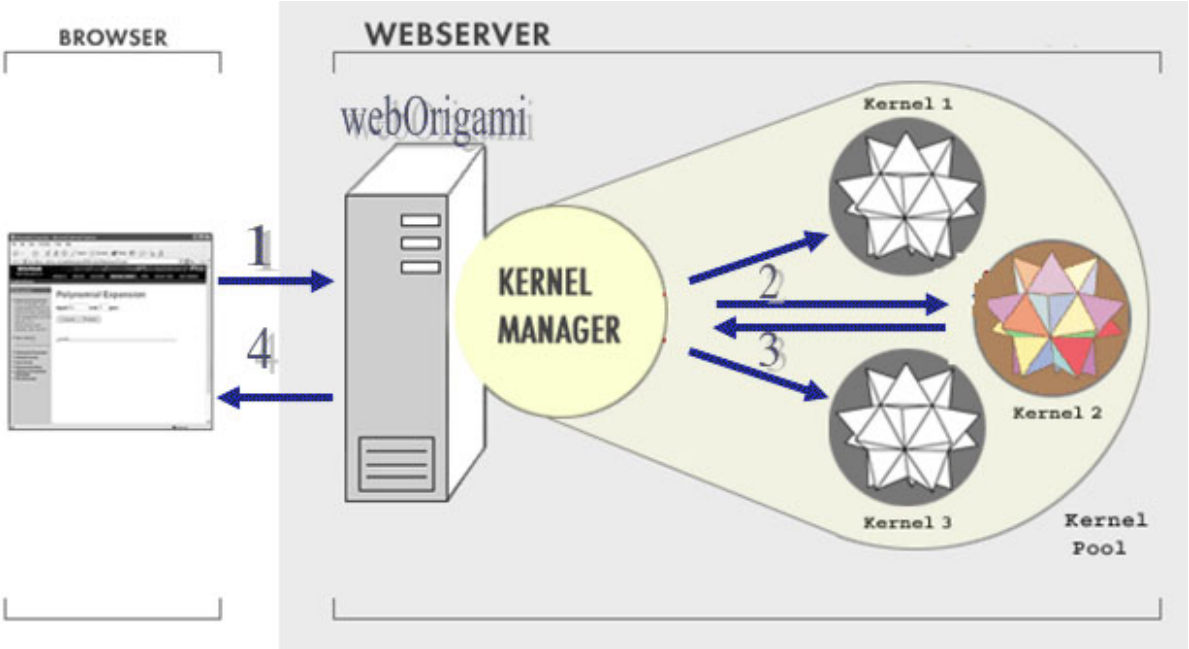


Figure 2: *webOrigami* simplified work flow

fold brings the backside of these faces to each other.

- 3D image view of the constructed origami.
- Usage of LiveGraphics3D that generates origami as a 3D Java object, which can be manipulated by user's mouse to perform 3D rotations. This requires the browser to be Java enabled in order to view 3D objects.
- Saving user's session work for the purpose of later use.
- Saving the construction as *Mathematica* notebook that includes user's specification for fold operations. This notebook is generated according to *Eos* syntax, and can be used to simulate the construction on *Mathematica* frontend and perform reasoning and theorem proving. This of course requires *Mathematica* installation and *Eos* packages.
- Viewing of pre-made origami pieces that helps in learning about construction steps.
- A set of useful functions like duplicating or deleting points, unfolding origami, turning it over, or rolling back to a previous step.
- Other viewing options and useful information that can be configured during construction steps.

Although our system can be classified as a client-server application, it requires some special processing on server side that traditional applications don't need to bother about. The reason for this is the origami structure and information that must be saved during all steps of construction. Every construction is performed in a context and data structures that shouldn't be overlapped with other's of concurrent constructions. The size of this required data is big enough not to be considered as classical session environment. Our implementation for *webOrigami* provides a transparent layer to separate users from each other using a username, or user identifier, without the need to modify *Eos* implementation. Thus, each construction process is carried out in private context allocated for the user on server side, and lasts during the session time of logged user.

Currently we are working on the integration of *AJAX* technology into *webOrigami*, which will al-

low asynchronous connection with the server and enhanced client interface.

5 Conclusion and Future Work

We have presented computational origami system, *Eos*, which not only simulates origami folds, but also computes and proves geometric properties of the construction. *Eos* keeps track of the geometrical properties during the construction phase, and then translates them into polynomial algebraic constraints which are supplied to theorem provers in the proving phase. We also explained about *webOrigami* project and its features.

Our experience with *Eos* shows that the number of polynomials grows very rapidly as the number of origami construction steps grows, which makes theorem provers run out of computing resources. These limitations can be overcome by adopting a computational framework with access to networked resources.

We are directing our research to grid computing as the web provides support for distributed theorem provers, which do not have the restrictions of one PC or workstation, and therefore they can cope with large systems of constraints.

Acknowledgements

This research is supported in part by the JSPS Grants-in-Aid for Scientific Research No. 17300004 and No. 17700025, and by MEXT Grant-in-Aid for Exploratory Research NO. 17650003. We also acknowledge the contribution of Dorin TEPENEU, a former PhD student in our laboratory, in the development of *webOrigami* system.

References

[1] R. Geretschläger. *Geometric Constructions in Origami*. Morikita Publishing Co., 2002. In Japanese, translation by Hidetoshi Fukagawa.

[2] T. Hull. Origami and geometric constructions. 2005. http://www.merrimack.edu/~thull/omfiles/geoconst.html.

[3] H. Huzita. Axiomatic Development of Origami Geometry. In H. Huzita ed, editor, *Proceedings of the First International Meeting of Origami Science and Technology*, pages 143–158.

[4] T. Ida, D. Ţepeneu, B. Buchberger, and J. Robu. Proving and Constraint Solving in Computational Origami. In *Proceeding of the 7th International Symposium on Artificial Intelligence and Symbolic Computation (AISC 2004)*, volume 3249 of *Lecture Notes in Artificial Intelligence*, pages 132 – 142, 2004.

[5] T. Ida, H. Takahashi, D. Ţepeneu, and M. Marin. Morley's Theorem Revisited through Computational Origami. In *Proceedings of the 7th International Mathematica Symposium (IMS 2005)*, 2005.

A . Appendix of Construction Code

Description

This note book shows a stepwise usage of *Eos* system to construct a regular heptagon using Huzita's folding axioms. And it demonstrates how to use the system for correctness proof of the construction.
To construct Origami shapes, we use some notational convention in this paper. We denote points by a simple capital letter A, B, C,..., possibly subscripted, a line passing through points X and Y by XY, and segments between points X and Y by XY.

Eos Initialization

```
<< OrigamiBasics`;
SetOptions[ShowOrigami, ShowFrame → True];
SetOptions[Fold, MarkPointOn → False];
$imageSize = 200;
```

Regular Heptagon Construction

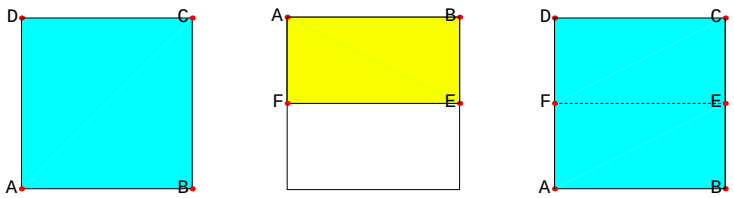
Construction steps

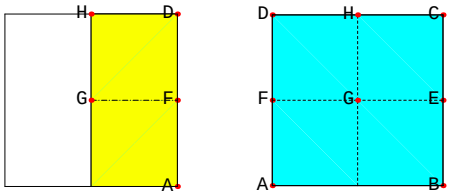
In order to obtain a regular heptagon, we base on Huzita's method that defines a set of axioms for origami construction. First, we need to obtain two important points; the center of heptagon and the first vertex. Then, we construct the second vertex. Finally, we can obtain the other vertices by line symmetries.

Construction of heptagon's center G and first vertex H

First, we define an origami object $ABCD$. Then, a crease that brings A onto D is computed. We name this crease EF. Now, we bring A onto B. G, the center of heptagon, is the intersection between the crease and segment EF. H, first vertex of heptagon, is generated as the intersection between the crease and segment DC.

```
BeginOrigami[{10, MarkPoints → {"A", "B", "C", "D"}}, FaceColor → {Hue[.5], Hue[.17]}];
Fold[A,D, MarkPointOn → True];
Unfold[];
Fold[A,B, MarkPointOn → {CD, FE}];
Unfold[];
```

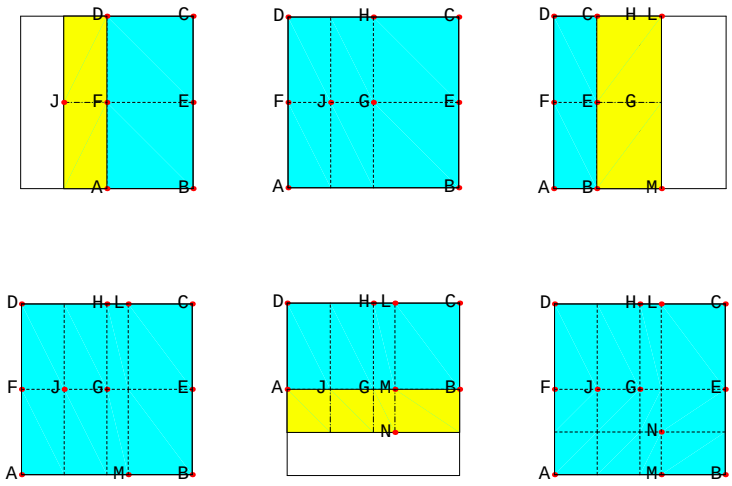




Construction of second vertex U

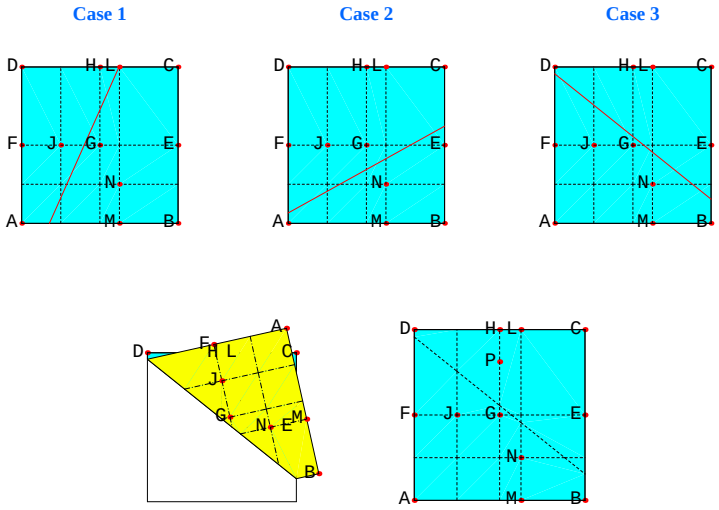
The point H is first vertex of regular heptagon. To prepare the construction of second vertex, we need to construct two important points J and N. J is the intersection between FE and the crease that brings F onto G. We name ML the crease that superpose E and J. Thus, N is the intersection between ML and the crease that brings A onto F.

```
Fold[F,G,MarkPointOn → {FE}];
Unfold[];
Fold[E,J,MarkPointOn → {AB, CD}];
Unfold[];
Fold[A,F,MarkPointOn → {ML}];
Unfold[];
```



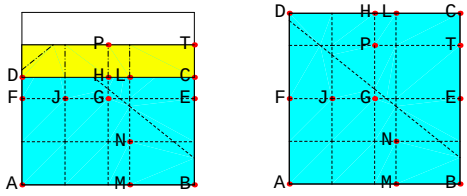
We make a fold to superpose point J to line GH and point N to line GE respectively. To find such fold is equivalent to solve a cubic equation (Huzita's 6th axiom) which can not be done by ruler-and-compass method [1, 2].
Since three fold lines are possible, we choose the third one. Then, we duplicate point J onto the line GH and we obtain the point P.

```
Fold[J, GH, N, GE];
Fold[J, HG, N, EG, Case → 3];
DupPoint["J"];
Unfold[];
```



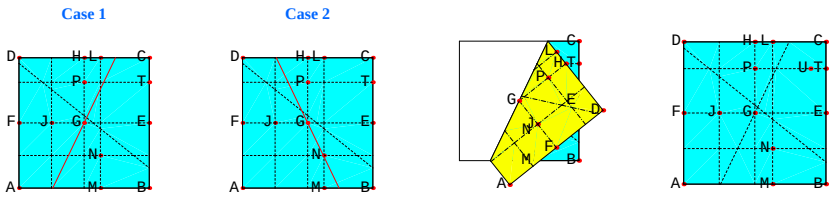
The second vertex of regular heptagon is on the perpendicular of GH passing through P.

`Fold[H, GH, Through → P, MarkPointOn → {BC}];`
`Unfold[];`



To obtain the second vertex, we make a fold along a line passing through G such that H is superposed on the line PT.
Since two folds are possible, we choose the first one. Then, we duplicate the point H onto the line PT and we obtain the second vertex U.

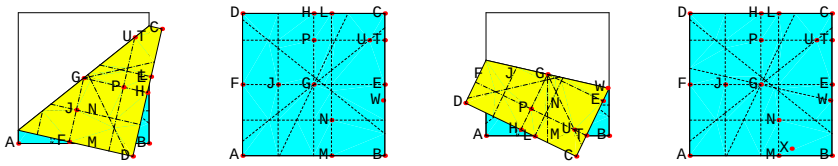
`Fold[H, PT, Through → G];`
`Fold[H, PT, Through → G, Case → 1];`
`DupPoint["H"];`
`Unfold[];`



Construction of other vertices W, X, F1, E1 and A1

We obtain the other vertices of the heptagon by line symmetries. The third vertex W is obtained as a symmetric point of H along the line UG. The fourth vertex X is also obtained as a symmetric point of U along the line WG.

```
Fold[H, Along → UG];
DupPoint["H"];
Unfold[];
Fold[U, Along → WG];
DupPoint["U"];
Unfold[];
```



The three remaining vertices *F1*, *E1* and *A1* are images of *U*, *W* and *X* respectively, by line symmetry with respect to *HG*. And finally we can obtain the regular heptagon as shown in the last step.

```
Fold[B, Along → HG];
DupPoint[{"U", "W", "X"}];
Unfold[];

ShowFolded[
Show → {ShowMarkPoints → {"A", "B", "C", "D", "G", "H", "A1", "E1", "F1", "X", "W", "U"},
More → Graphics3D[{Thickness[0.02], Hue[0], GraphicsLine[{H,A1,E1,F1,X,W,U}]}]}];
```