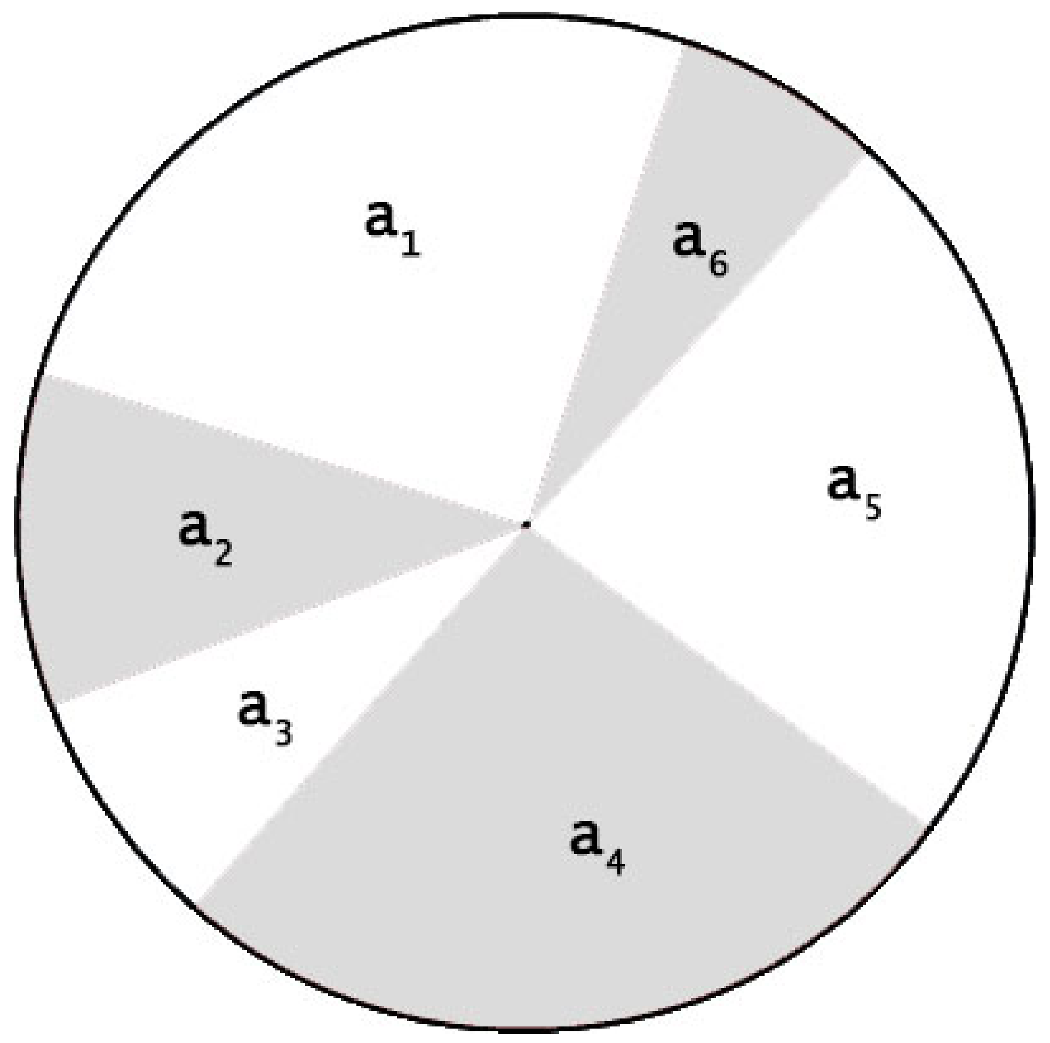
Origami Simulation

Abstract

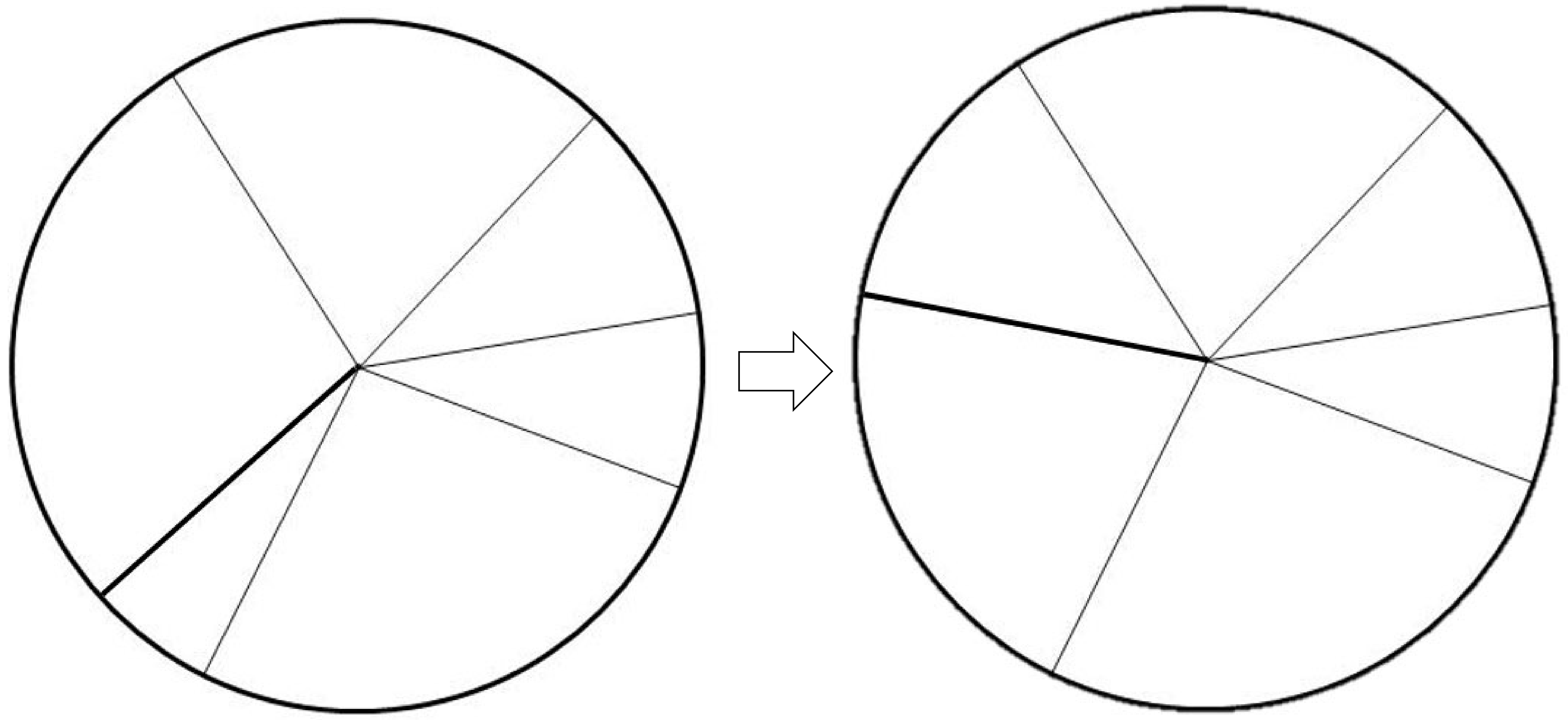
There are many open-ended problems and proofs to describe the math behind origami, and even more opportunities to simulate these untold rules of folding. This project is an adventure in programming, origami foundations, and simulation. It allows users to create single-vertex crease patterns, make them foldable, and save the image in order to print and physically interact with it.

Much inspiration came from Erik Demaine's lectures, which combine computational algorithms with interesting origami applications. I also came across Robert Lang's Treemaker and Jun Mitani's ORIPA during research, and although their functions are not as closely linked with this project, the programs contribute to the same knowledge base.

I focused on modeling one particular aspect: Kawasaki's Theorem. It states that the consecutive alternating sum of the angles formed by creases emanating from a single vertex should equal zero. In visual mathematical terms:



$$a_1+a_3+a_5 = a_2+a_4+a_6 = 180^\circ$$
$$a_1-a_2+a_3-a_4+a_5-a_6 = 0^\circ$$



Design

The original idea started out with an inkling of Java, origami aspirations, and a 6 month timeframe. I began when I was halfway through an introductory computer science course. Continual research provided each next step to take; I learned to use the Eclipse IDE, became familiar with the SWT components for user-interface capabilities, and began to understand what defines a good developer.

The first part of project development was creating the GUI components. This was particularly frustrating because I had no experience with front-end work, much less SWT. It was necessary, though, since I didn't want to start the algorithms without a visual way to test and check them. The menus, buttons, labels, canvas, mouse listeners, and paint events were all implemented, most with the assistance of the Window Builder plugin.

The second part of development focused on manipulating the data inputs. The program tracks the mouse over the canvas region. Each click stores a corresponding point on the circumference of the circle, a pixel with x-y coordinates. This point has an associated angle calculated based on its position. Custom math methods accomplish these tasks.

Example console output:

```
Angles between lines:
74.41924026339427
134.88076921456764
105.58075973660573
45.119230785432364
Sum of alternating half: 180.0
```

One algorithm checks if the current crease pattern would be foldable by adding the alternating angles.

Another modifies the folds to make the pattern foldable by finding one alternating sum, finding the difference necessary to shift in order to become 180 degrees, finding the largest angle, and decreasing that angle by the difference. There is another case for this algorithm when there are an odd number of folds present, and more folds must be created or rearranged in the proper position to partition angles properly and satisfy the conditions.

The featured theorem is not new; in fact, the concept was initially discovered in the late 1970's, and it has been developed and improved upon through the 1990's.

However, this simulation provides an interactive experience to explore the abstract relationship between input and output. The user can both create creases virtually and save the canvas region as an image, which allows for printing and physical folding.

Conclusions

The unexplored nature of origami and its possibilities for simulation provide ample opportunity to expand this project in the future. To add on to flat-folding, I could feature mountain/valley assignment (Maekawa's Theorem) or local foldability among multiple vertices. There are hopes that in the future, I can contribute to the rigorous studies in origami, flat-foldability, and the mathematics behind it all. There are also design features that I could add, such as saving as different file types, opening files, and refining the user experience.

I learned that a self-contained software program requires more attention to detail, consistency, and organization than independent algorithms and code snippets. This project and its required problem-solving taught more about software design than any reading material or lecture could, even if it was a small-scale individual project.

References:

6.849: Geometric Folding Algorithms: Linkages, Origami, Polyhedra (Fall 2010). (2010, Fall). Retrieved May 11, 2017, from http://courses.csail.mit.edu/6.849/fall10/lectures/

Mitani, Jun. ORIPA: Origami Pattern Editor [Computer Software]. (Ver.0.35). Retrieved from http://mitani.cs.tsukuba.ac.jp/oripa/

Contact:

jkwongfl@yahoo.com

github.com/wong-justin