FRAMESHIFT: SHIFT YOUR ATTENTION, SHIFT THE STORY

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Master of Science
in
Computer Science with a Concentration in Digital Arts

by

Tim Tregubov

in Conjunction with Rukmini Goswami

Department of Computer Science

DARTMOUTH COLLEGE

Hanover, New Hampshire

May 15, 2015

Examining Committee:

Chair _____
Lorie Loeb

Member _____
Michael Cohen

Member _____
Michael Casey

_____
F. Jon Kull, Ph.D.
Dean of Graduate Studies

FRAMESHIFT: SHIFT YOUR ATTENTION, SHIFT THE STORY

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Master of Science
in
Computer Science with a Concentration in Digital Arts

by

Rukmini Goswami

in Conjunction with Tim Tregubov

Department of Computer Science

DARTMOUTH COLLEGE

Hanover, New Hampshire

May 15, 2015

Examining Committee:

Chair _____
Lorie Loeb

Member _____
Michael Cohen

Member _____
Michael Casey

_____
F. Jon Kull, Ph.D.
Dean of Graduate Studies

ABSTRACT

INSERT ABSTRACT HERE

Acknowledgements

To our

To our families

Contents

List of Tables

List of Figures

Chapter 1

Design Contributions

I. Design Philosophy

modularity

friendliness

delight

II. Interface Iteration

TODO: include preliminary user study 1

sfdaf

III. Interactions

TODO: add tap options, how to draw, and a description of the tool-based interface in general

(a) Tutorial

We eschewed detailed drawing instructions or a separate tutorial mode, in favor of short video tutorials that appear the first time each tool is used. These tutorials can also be accessed by tapping the feature icons on the about page.

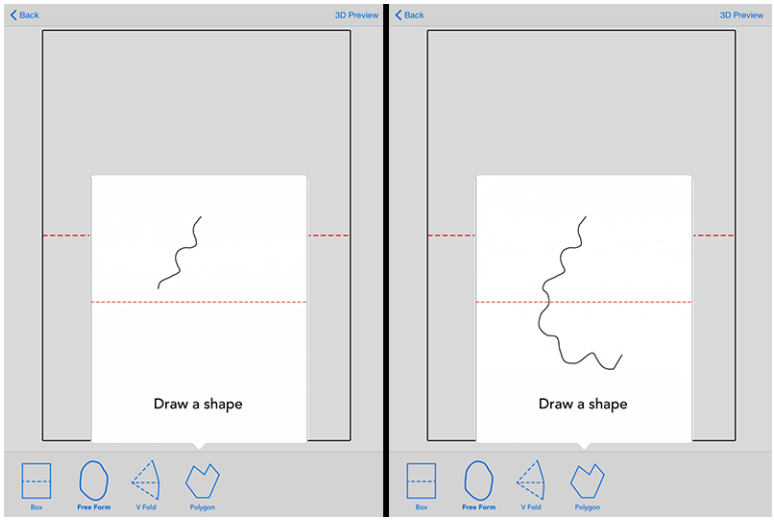

Figure 1.1: Free-form shape tutorial video.

We also show helpful tips between screens — for example, when moving to 3D preview and restoring from a saved sketch.

(b) Tool Interactions

Some interactions are common to all features. To add a feature

Box Fold

To draw a

FreeForm

Polygon

V-Fold

(c) Warnings and Errors

A central . We display warnings and error as

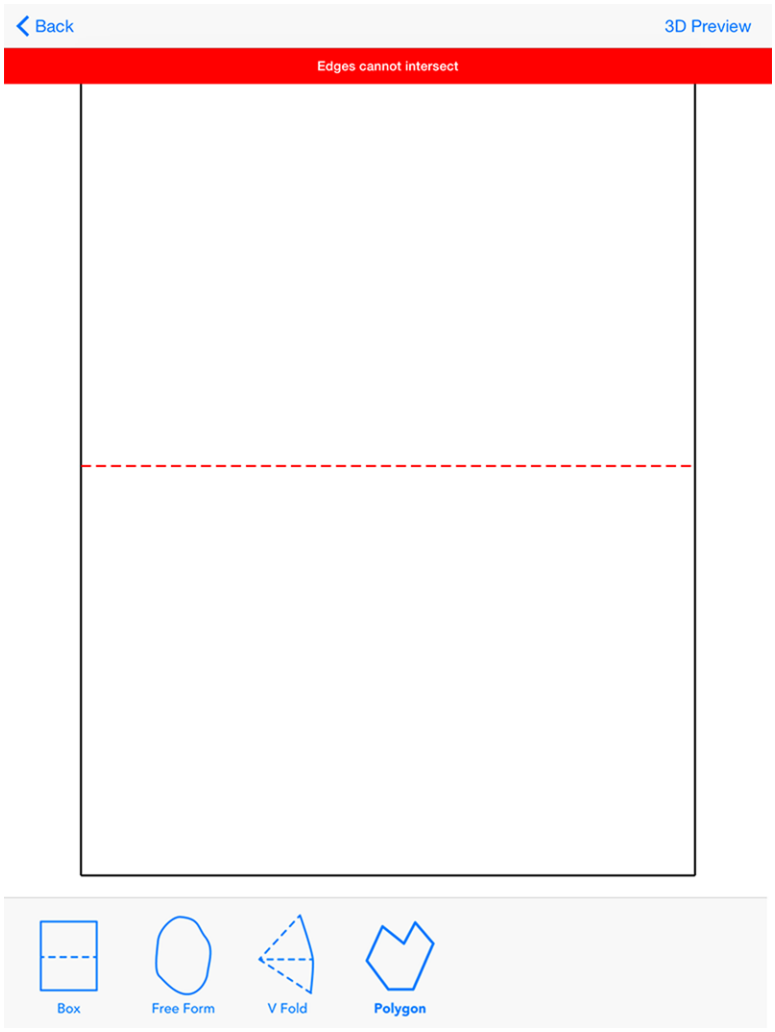

Figure 1.2: An error message shown when rejecting a polygon with intersecting edges.

(d) Send to Laser Cutter

In the three-dimensional preview,

(e) Print

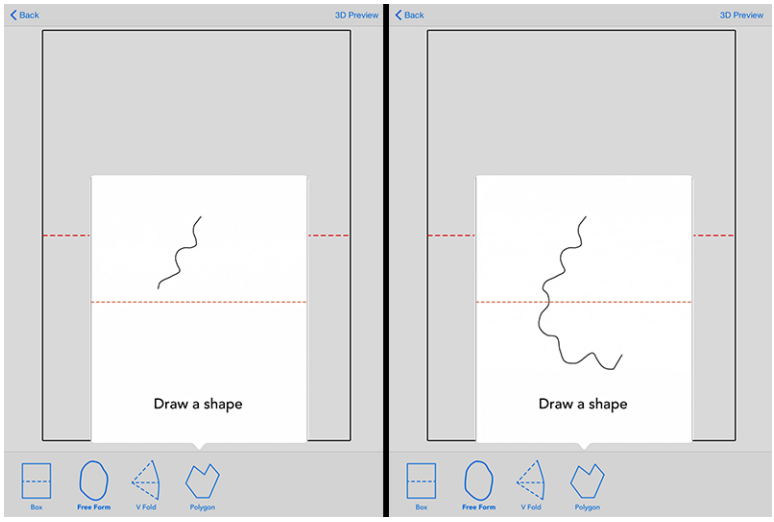

Figure 1.3: Options for sharing a fold pattern from the 3D preview.

IV. Interface Data Structures

We will refer to several data structures throughout the discussion of user interface design and implementation. These are the primary means of storing user input processed by the tools, and are processed by our algorithms to draw designs in 2D and simulate them 3D[1].

(a) Edges

An Edge represents a cut or fold. Edges are the basic building block of planes, and an integral element of all fold features. An edge is minimally defined by a start point, end point, and a a type (either cut or fold). This minimal definition represents a straight edge between two points. In addition, an Edge can contain further information: the bezier path drawn to create it (for non-straight edges), and a reference to the plane or feature it is a part of. Additionally, each edge contains a reference to its "twin" edge.

Twin Edges

Although it is often simplest to think of edges as cuts and folds created by the user, the reality in Foldlings is slightly more complicated. For each edge that the user creates using a tool, two edges are created. We create edges with direction, such that there is an edge from the start point to the end point of the edge, and another edge starts at the endpoint and has the reverse path of the original edge. This distinction is most important when detecting planes from edges[2], but must also be taken into account whenever edges are processed. For example, when drawing the 2D view of a sketch, we skip drawing the twins of edges already drawn, which reduces drawing work by half.

[1] This section only describes the primary data structures necessary for constructing fold and patterns from user input, detecting planes, and determining the relationships between, not the systems for drawing features in 2D or 3D. For a discussion of 2D drawing, see sections. For a discussion of, see . **>>TODO:cite marissa >>TODO reference section**

[2] **>>TODO cite marissa's planes**

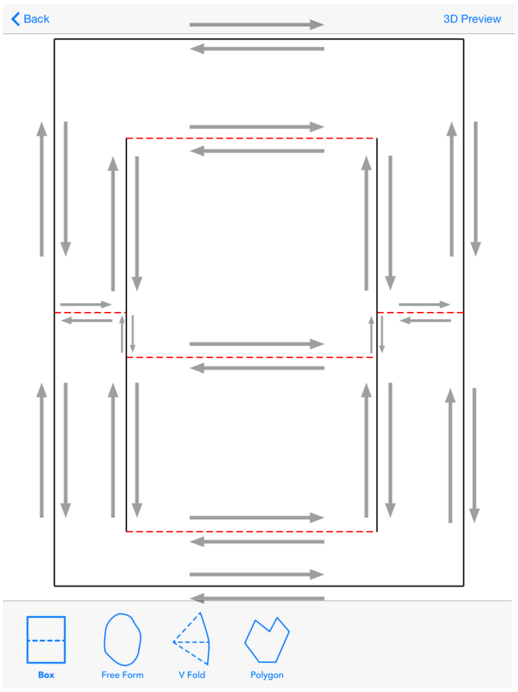

Figure 1.4: This sketch contains 34 edges, with orientations shown by the overlaid gray arrows.

Driving Folds

A driving fold is not a special type of edge, but rather a relationship between an edge in one feature and a feature "spanning" that edge. A feature is said to span a fold when it is drawn on top of an existing fold, so that it has horizontal folds on both sides of the middle fold.

An edge can be the driving fold for more than one feature, but each feature has only one driving fold (if there are multiple potential driving edges at the same height, the leftmost edge is selected. The driving fold is important for calculating parent-child relationships between features: a feature's parent is the feature that contains it's driving fold[3]. These parent-child relationships are described in more detail in the Nested Features section on page 18.

[3]The exception to this rule is holes — a hole's parent is the feature that contains it.

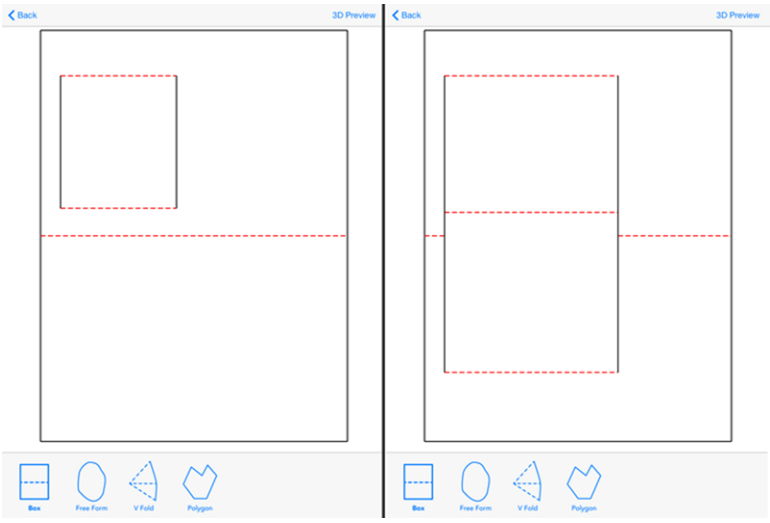

Figure 1.5: Left: a box fold mid-drag. The feature does not have a driving fold. Right: a box-fold after the user has released the touch. The feature's driving fold is the master card's middle horizontal fold.

Fold Orientation

Traditionally, kirigami patterns are based on. **>>TODO: FINISH SECTION, cite >> TODO KIRIGAMI PATTERN FIGURE**

(b) Planes

Planes are an enclosed shape, bounded by edges. Plane are detected from edges by traversing the directed edge graph, as described in **>>TODO: CITE MARISSA HERE**. They are drawn as colored areas in the two-dimensional sketch, and simulated in the 3D preview as shapes that rotate about a pivot point. In order to simulate the planes in 3D, we construct parent-child relationships between the planes, which determine how they move during simulation.

(c) Fold Features

The central data structure of Foldlings is the FoldFeature: a representation of a shape drawn by the user that folds in 3d. Each fold feature is a single design element — and can be indi-

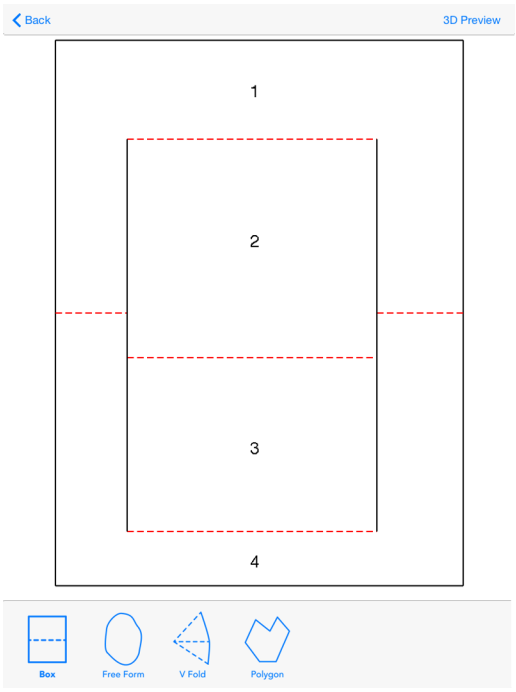

Figure 1.6: Planes in a simple sketch, numbered by ancestry. Starting at the root plane 1, each successive plane is the child of the previous numbered plane.

vidually created, modified, and deleted. There are five classes of FoldFeature: MasterCard, BoxFold, FreeForm, Polygon, and V-Fold, representing differences in drawing behavior, geometry, and (the differences are described in detail below). Each of these features is a subclass of the FoldFeature superclass.

All FoldFeatures have functionality in common:

- Each feature contains a list of edges in the feature — cuts and folds, including twins.
- Each feature has a driving fold — in the case of unconnected features, such as the master card and holes, the driving fold is nil.
- Each feature can be deleted from the Sketch, "healing" the sketch by closing gaps left in any
- Features implement the encodeWithCoder and decodeWithCoder methods, allowing them to be serialized to a file on the device and restored from the saved file.
- Each feature can provide a list of current "tap options" — actions that can be per-

formed on the feature given its state. **>>TODO:SEE tap options in interface design**

- Each feature can perform hit-testing: given a point, it can determine whether that point is inside or outside the feature. [**?**]. **>>TODO:REMOVE CITATION – JUST TESTING**

Master Card

Each sketch always contains a single master feature, which is the ancestor of all other features. It is a simplification of the box fold, in that it contains three horizontal folds with connecting vertical cuts. Users do not create features of this type — each sketch begins with one. All of the edges in the master feature are marked with a flag indicating that they belong to the master feature, because master feature edges and planes are sometimes treated differently than normal edges. For example, the parent-child relationships between planes are constructed by starting at the top plane in the master feature, determined by edge type and height.

Box Fold

A box fold consists entirely of straight edges, and can be constructed from two points: the top left point, and the bottom right. The middle fold position is determined by the position of the driving fold. Box folds are only valid if they have a driving fold.

Free Form

Free-form shapes are defined by a single, closed path. When the feature is completed (by releasing the touch), the shape is truncated, horizontal folds are added, and the path is split into multiple edges (assuming the shape spanned a fold)[4]. The curved path is defined by a set of "interpolation points" — points captured by sampling touch positions while a user

[4] **>>TODO:SEE SECTION**

draws a shape on the screen. A bezier path is interpolated between these points using the Catmull-Romm algorithm **TODO: CITE**.

Holes are a special case of FreeForm shapes, and are cut out from the final design, rather than simulated as a separate plane. FreeForm shapes that do not cross a fold are considered holes — drawn in white in the 2d sketch and drawn as subtractions from planes in the 3d view.

Polygon

Polygons are created from a list of "tap points" constructed from user input. As in free-form shapes, these points are connected with a bezier path, and are truncated if they contain a driving fold when they are completed. For polygons, this path consists only of straight line segments. Unlike interpolation points, tap points can be moved at any time during the drawing process.

Like FreeForm shapes, Polygons that do not have a driving fold are considered holes.

V-Fold

V-Folds are defined by a path that crosses the driving fold, called a "vertical cut." This path can be an arbitrary shape that crosses the driving fold once. They are fully-defined by adding a point on the driving fold. From this point, we construct three diagonal folds, two to the top and bottom of the vertical cut, and one to a point that intersects with the vertical cut at a point calculated to make a valid 90-degree feature.

V-folds are only valid if they have a driving fold, and their vertical cut intersects the driving fold exactly once.

(d) Sketches

A sketch is the representation of the user's drawing — its primary role is as a collection of features. It also contains information about the current drawing state, and the state of

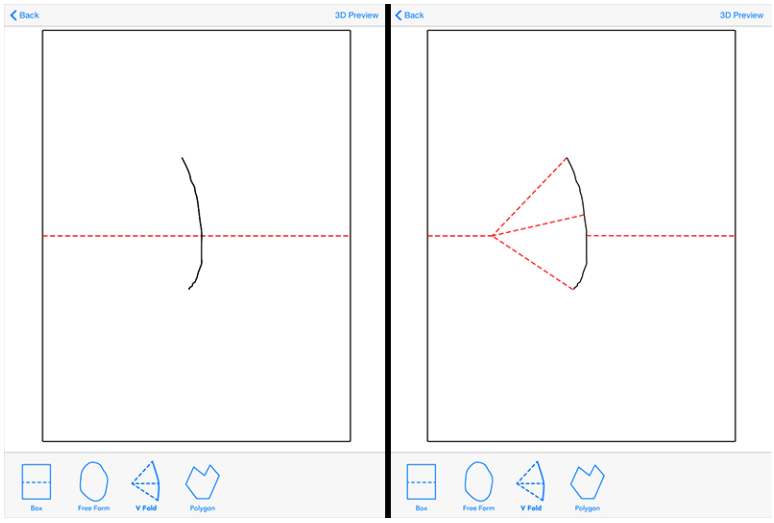

Figure 1.7: Left: an unfinished v-fold, consisting only of a vertical cut. Right: a v-fold after defining the point on the driving fold to create diagonal cuts.

user interaction (for example, which features are currently being modified, and which tool is selected).

V. Saving

afsdfs dsf sf

VI. User Interface Future Work

(a) Concurrency

(b) Multiple Cards

Chapter 2

Technical Contributions

I. Interface Implementation

sfds

II. Tool Implementation

```
var horizontalFolds:[Edge] = [] //list horizontal folds
var featureEdges:[Edge]?        //edges in a feature
var children:[FoldFeature] = [] // children of feature
var drivingFold:Edge? // driving fold of feature
var parent:FoldFeature? // parent of feature
var startPoint:CGPoint?
var endPoint:CGPoint? // start and end touch points

/// splits an edge around the current feature
func splitFoldByOcclusion(edge:Edge) -> [Edge]
{
//by default, return edge whole
return [edge]
}
/// features are leaves if they don't have children
func isLeaf() -> Bool
{
return children.count == 0
}
/// options or modifications that can be made to the current feature
func tapOptions() -> [FeatureOption]?
{
  return [FeatureOption.PrintPlanes, FeatureOption.PrintEdges,
  FeatureOption.ColorPlaneEdges, FeatureOption.PrintSinglePlane]
}
/// whether a feature is drawn over a fold, determines whether
/// a fold can be the driving fold for a feature
  func featureSpansFold(fold:Edge)->Bool
{
  return false
}
/// returns and calculates planes in a feature
func getFeaturePlanes()-> [Plane]{
  return featurePlanes
}
/// whether a feature contains a point
/// needs to be overridden by subclasses
func containsPoint(point:CGPoint) -> Bool{
  return self.boundingBox()?.contains(point) ?? false
}
/// returns edges that are less than the minimum length
func tooShortEdges() -> [Edge]{
```

```
  return featureEdges?.filter({$0.length() < kMinLineLength}) ?? []
}
/// this function should try to fix errors first,
/// then return an error if it can't fix them
/// returns whether the feature was valid
func validate() -> (passed:Bool,error:String){
  var valid = true
  if(!tooShortEdges().filter({$0.kind == Edge.Kind.Fold}).isEmpty){
  return (false,"Edges too short")
  }
  println("valid")
  return (true,"")
}
```

III. Nested Features

a

IV. Deletion

a

V. Intersections Between Features

sfsfs

VI. Validity

One for the

```
override func validate() -> (passed: Bool, error: String) {
 let validity = super.validate()
 if(!validity.passed){
    return validity
 }
 // clever test for concave paths: close the vertical cut's
 // path and test whether vfold end point is inside it
 var testPath = UIBezierPath(CGPath: verticalCut.path.CGPath)
 testPath.closePath()
 if(testPath.containsPoint(diagonalFolds[0].end)){
    return (false,"Angle too shallow")
 }
 if(!tooShortEdges().filter({
    \$0.kind == Edge.Kind.Fold
    }).isEmpty){
        return (false,"Edges too short")
 }
    return (true,"")
 }
```

VII. Future Work

fdjas

Chapter 3

User Studies

I. User Test at the Digital Arts Exhibition

On April 28, 2015, we tested our system with attendees of the Digital Arts Exhibition at Dartmouth. After a brief demonstration of how to create folds and preview their design, users designed cards using Foldlings. Users drew sketches and then sent an email containing an SVG file to the computer connected to the laser cutter. Finally, they placed a piece of paper in the laser cutter, and watched as the laser beam cut out their design. Over the course of two hours, users cut and folded 31 popup cards. <<TOOO: CITE DAX <<TODO:

The system we demonstrated at the exhibition was incomplete — it contained the basic box fold and freeform shape tools, but did not include some advanced features of the final software, such as dragging folds or shading based on plane orientation. The alpha software also contained several bugs that disrupted the experience. However, the system was usable enough for people to create cards, and observing user behavior was invaluable in designing our final product.

Because users were new to our system — and constrained by the pressure of other users waiting to design cards — designs were relatively simple. Sketches generally contained between 2 and 5 fold features in addition to the base card — the most complex design con-

tained 10 fold features. Despite their simplicity, sketches showed a wide range of designs, ranging from abstract shapes to representational scenes — users sketched symbols, Chinese characters, and geometric forms. Most of the sketches utilized both freeform and box fold features, mixing the two element types to create a composition. One of the most popular design elements was the user's name: 5 of the cards contained names or initials. Roughly a third of the designs took advantage of nesting — constructing fold features inside each other.

Because users were able to quickly design and fabricate their design, people generally left satisfied. People typically spent around 20 minutes at our booth, leaving with a popup card they had created. However, the experience was not frictionless. Users were frustrated by crashes: touching the screen with more than one finger or drawing while calculating planes were the most common reasons for failure. Other common complaints were the lack of a delete/undo button and that the UI did not show which tool was currently selected.

Folding the fabricated design also presented difficulties. Although they were able to see a 3d preview of their design while creating it, users had often relinquished the iPad by the time they folded their design. They were often unsure how to fold their card, and struggled to discover the correct fold orientations. In some cases, it took longer for users to fold their creation than to design it.

We observed several unexpected behaviors. A few users rotated the screen to design a card in a landscape view, rather than the portrait orientation implied by the orientation of the buttons and 3d preview. They used this orientation to design cards that folded medially rather than laterally. Several users also constructed overlapping features by drawing on top of existing features. These features did not simulate correctly, as they intersected with existing edges. However, this behavior demonstrated a desire to construct more complex geometry. In the final software we implement unions for fold features — the most recently-drawn feature occludes features underneath it, modifying their edges.

Users also relied on the 3d preview to differing degrees. Some users viewed the preview

after every operation, while others only switched to the preview occasionally. Many users relied on the 3d preview as a reference to how to fold their popup card. We were surprised by this, and conducted further user studies to determine the effectiveness of methods of displaying 3d information.

II. Visual Aids User Study

goal: test whether users understand the mapping of 2d fold patterns to 3d, and test the degree to which plane coloring, edge patterning, and a 3d preview help users understand how a popup card will fold.

Since. >>TODO: cite other research on visual aids for folding slash other 2d to 3d vis

(a) Method

Each subject received a set of five laser cut cards, and we recorded the time it takes them to successfully fold the card.

For each card, each subject was randomly given one of of the following five aids:

1) A two-dimensional design, showing planes shaded by whether they will be horizontal or vertical when folded.
2) A two-dimensional design, with edges patterned based on whether they are "hills" or "valleys" — whether they fold towards or away from the card.
3) A video showing a simulation of the card folding in three dimensions.
4) A still image of the card folded in dimension.
5) No visual aid.

The order of aids and cards was shuffled, and then balanced to ensure an equal distribution of orderings. Eg. each visual aid has an equal chance of being the first aid presented to a user and

Finally, we asked subjects to rank the visual aids

The effect each type of aid has on folding time will help determine which types of visualization to include in Foldlings

(b) Results and Discussion

III. Final User Study

As a final test of software, we compared the usability with Foldlings to traditional manual popup design methods. First, we presented participants with a completed popup card, and asked them to replicate the design using manual cutting and folding and to create the design using our software. The order was randomized, so half of participants were given were given the manual first, and half started with Foldlings. We timed how long the participants took to complete the design. Next, we gave them a free-form exercise: make a design that includes a tree." Finally, we asked participants to rate the experience of using our tool and their satisfaction with the popup card they produced. >>TODO: attach image >>TODO graphs, data, participants, do the study lol

>>TODO: update