Creating Directions for Origami using HTML5

Jeanine Meyer
jeanine.meyer@purchase.edu

Abstract: Features in the current specification of HyperText Markup Language and JavaScript, termed HTML5, namely drawing on a canvas and incorporating video, provide powerful and flexible capabilities for preparing directions for folding origami models. The methodology shown here demonstrates topics in geometry, trigonometry and algebra as well as basic programming that would appeal to teachers and students. In this paper, after providing background on origami and on HTML5 and JavaScript, I describe the programs for the business card frog and the talking fish. At a later stage, because Processing is the programming language we use for our CS1 class, I produced a Processing sketch for the talking fish.

1 Background

As participants in this conference know, origami refers to the art of paper folding. It is typically associated with Japan, but has roots in China and Spain as well. Traditional folds include the water bomb and the crane and flapping bird. Lillian Oppenheimer is credited with popularizing origami in the United States and started the organization that became the American national organization (see OrigamiUSA). She personally taught me the business card frog in 1972. Origami is a vibrant art form practiced around the world as well as the focus of research in mathematics, engineering, and computational complexity.

There is a standard format for book directions, aka diagrams, for origami and I built on that standard. In this approach, each step shows the next fold to be made using a set typography. The most basic folds either assume a valley shape when unfolded or a mountain shape and this is indicated by dashed or dotted and dashed lines. Any subsequent steps with folds unfolded typically indicate the folds using thin lines unless the sense (valley or mountain) is critical for making the next fold.
As will be demonstrated by looking at the directions and reading the next sections, my methodology provides a way to achieve exact positions for the line drawings since all coordinates are calculated. It also makes use of images and videos for operations for which line drawings may not be good enough for the reader.

Hypertext Markup Language 5 (HTML5) is the latest specification for documents on the Web (see W3C). Most browsers recognize most of the features. These programs were tested in Firefox and Chrome. Safari and Opera probably work. Many books (see Meyer) and online sources exist for learning HTML5.

The specification of HTML5 includes specification for the application development interface (API) for JavaScript and other languages. The bulk of these directions are in the JavaScript script as opposed to HTML.

HTML5 provides a mechanism called the canvas element on which the programmer can write code to draw paths (lines and arcs) and rectangles as well as images using image files. These operations are performed using a coordinate system to specify positions. The coordinate system has the property that the origin is in the upper left corner of the canvas and vertical values increase moving down the screen. This is the opposite of the usual approach in which the origin is at the lower right and vertical values increase going up the page. I could have finessed this by using the transformations provided, but I chose to accept it as is since it is common in computer systems. JavaScript includes a library of mathematical functions provided in what is called the Math class. This was useful for the trig functions and square root.

HTML5 provides native support of video, meaning that no special plug-ins are required, nor is it necessary to link to YouTube or other video hosting sites. A negative feature is that the browser makers have not agreed on a single encoding (codec) for video so that the practice is to create 3 versions of each video clip, with each browser using the one it supports. This situation may improve.

When the HTML5 specification was first announced, I wanted to find applications that made use of the new features and this led me to the production of origami directions.

HTML5 and JavaScript provide many ways for interaction. In my origami applications, the web page has buttons so the viewer/folder can advance or go back to see the steps of the directions.

Readers are invited to try out my directions by going to the websites: http://faculty.purchase.edu/jeanine.meyer/html5/origamifrog.html and http://faculty.purchase.edu/jeanine.meyer/html5/origamifish.html

The source code for HTML documents can be examined using commands in most browsers (e.g, in Firefox: View/Page source).
Origami Directions

Processing is a language built in and on Java that facilitates the production of interactive, graphical programs. Processing is a compiled language. The source code for the talking fish is available upon request.

2 Implementation

I describe the basic methodology I used to produce the directions, the utility functions required for these two examples, and then details for the frog and then the fish. I then describe how I produced the Processing sketch using the JavaScript code.

2.1 Basic Outline for HTML implementation

An HTML document consists of text and tags, the tags called markup. An example of a tag is <html>. Most tags come in pairs, with the second one bearing the same name preceded by a slash. The general structure is

<html>
<head>
<title>title appearing a special place in the browser</title>
<style> formatting directives </style>
<script> Javascript code </script>
</head>
<body>
content of the document, often structured using markup
</body>
</html>

A canvas element (there can be more than one, but my examples each just use one) is contained in the body. Similarly, a place, called a div, for the text describing each step is in the body and will be shown later in this section.

My aim was to produce line drawings, similar to those found in books, with calculations for the coordinate positions of the critical points and lines. That is, I did not want to measure and record lengths or angles, but have JavaScript do that task for me. This would work even for folds done 'to taste' as the origami jargon goes because I could determine the exact positions I chose to use.

JavaScript, just like other programming languages, has functions, pieces of code that can be invoked by name with or without extra information stored in parameters, and variables, names associated with values. My programs include many of what I call utility functions, for example, a function that draws a dashed line of an indicated color from a point defined by x1, y1 to a point defined by x2,
I also programmed functions for determining the intersection point of two lines and the distance between two points. I used variables to represent the coordinates of significant points of my model and other values. For example, the fish model is made with a standard sheet of origami paper, called kami. I defined a variable:

```javascript
var kamiw = 4
```

to hold the fact that the kami was 4 inches wide. I defined another variable representing the conversion from inches to pixels:

```javascript
var i2p = 72;
```

I decided that, after showing the representation of valley and mountain folks, the fish diagrams would start with the kami rotated to be a diamond shape with corners a, b, c, and d. The a corner is to the left, the b at the top, the c at the right and the d on the bottom, directly below the b. I needed to make one independent decision: determine the location in terms of pixels for a. From that point on, everything else is determined by calculations. The following variable declaration statements set up the values that will be used for all that follow:

```javascript
var diag = kamiw* Math.sqrt(2.0)*i2p;
var ax = 10;
var ay = 220;
var bx = ax+ .5*diag;
var by = ay - .5*diag;
var cx = ax + diag;
var cy = ay;
var dx = bx;
var dy = ay + .5*diag;
```

A similar approach is taken for the frog. Some calculations are more intricate than others.

Each folding step would be represented by a function to produce the image along with a piece of text. The image is produced on the one canvas element and the text placed in a div element. The entire <body> element for the frog follows:

```html
<body onload="init();">
<canvas id="canvas" width="900" height="400">
Your browser does not recognize the canvas element
</canvas>
<br/>
<div id="directions"> Press buttons to advance or go back </div>
<hr/>
<button onClick="goback();" style="color: #F00">Go back</button>
```
The `init` function starts things off by setting key variables and showing the first image with its accompanying text. The whole set of directions is specified using an array of arrays, `steps`, with each inner array holding the name of a function name and a string of text. This made it easy for me to develop each application by adding steps one at a time. I determined the functions and relevant positions. I also inserted new steps in-between as needed. In fact, I added the very first step/picture for each model after I had completed everything else. I decided it was important to display the conventions for valley and mountain folds.

The features of HTML5 made it possible to create a function for displaying a video clip or a photograph as easily as creating a function for drawing lines on the canvas. This meant that I did not need to change the overall mechanism when I decided to add a photograph for the last step of the frog and later insert video clips and photos at certain points for the fish.

The viewer (folder) uses buttons to go forward or backwards in the display of steps.

Figure 1 shows an intermediate step for folding the frog, accompanying text, and the buttons.
For purposes of this exposition, I will describe the utility functions next. Keep in mind that this was not the order in which they were created. After creating functions for representing valley and mountain folds, I started on the frog and did what was needed to describe each step.
2.2. Utility Functions

I define a function as a utility function if it may be of use to another origami model, though I developed the functions as needed when programming the frog directions and then the fish. The current set of functions is shown in the table.

<table>
<thead>
<tr>
<th>Functions relating to general scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>donext</td>
</tr>
<tr>
<td>goback</td>
</tr>
<tr>
<td>init</td>
</tr>
<tr>
<td>directions</td>
</tr>
</tbody>
</table>

### Diagramming

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>valley</td>
<td>draws dashed line, sometimes in a color, representing valley fold</td>
</tr>
<tr>
<td>mountain</td>
<td>draws dots-and-dashes, sometimes in a color, representing mountain fold</td>
</tr>
<tr>
<td>solidline</td>
<td>draws solid line (lineWidth is 2). Note: lines are drawn in other functions with the set lineWidth.</td>
</tr>
<tr>
<td>skinnyline</td>
<td>draws thin line (lineWidth is 1)</td>
</tr>
<tr>
<td>dot</td>
<td>draws dot, used by mountain function and debugging</td>
</tr>
<tr>
<td>shortdownarrow</td>
<td>draws small vertical arrows</td>
</tr>
<tr>
<td>curvedarrow</td>
<td>draws curved arrow</td>
</tr>
</tbody>
</table>

### Mathematics

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uselawofsines*</td>
<td>uses Law of Sines in specific way to calculate a position for frog</td>
</tr>
<tr>
<td>proportion</td>
<td>general function: returns position along line segment a specified proportion</td>
</tr>
<tr>
<td>dist</td>
<td>distance between two points</td>
</tr>
<tr>
<td>intersect</td>
<td>location of intersection of two line segments (assumes lines do intersect)</td>
</tr>
</tbody>
</table>

*The uselawofsines function is specific to the frog, but I decided to keep it around to remind me of how the Law of Sines can be used to determine an angle or a side length in a triangle assuming some information is known.*
As an example, I developed the code for intersect by determining the equation for each of the two lines and then writing the expression for the solution. The so-called two point equation for a line going through points x1,y1 and x2,y2 is
\[ y-y_1 = \frac{(y_2 - y_1)}{(x_2-x_1)} \times (x - x_1) \]
and the slope of this line is \( \frac{y_2-y_1}{x_2-x_1} \).

Using the same form, the equation for a line going through x3,y3 and x4,y4 is
\[ y-y_3 = \frac{(y_4 - y_3)}{(x_4-x_3)} \times (x - x_3) \]
and the slope of this line is \( \frac{y_4-y_3}{x_4-x_3} \).

The intersection is found by solving each of these two equations for y and then setting the two equal to one another. This produces an equation in x. I/my code then solve this equation for x. Lastly, I/my code uses one of the two equations to get the corresponding y.

I create new variables to simplify the expressions: \( m_{12} \) is set to the slope of the first line segment and \( m_{34} \) is set to the slope of the second line segment. The variable \( m \) is set to \( \frac{m_{34}}{m_{12}} \). Note that the calculation is to determine the intersection of the line segment going from (x1,y1) to (x2,y2) and the line segment (x3,y3) to (x4, y4). I am not consistent with naming: you can see in the examples below that I use \( midx \) and \( midy \) to define a position. Note also the comment in this code. This is not a general function, which would require checking for vertical or horizontal lines and for the lines being parallel.

```javascript
function intersect(x1,y1,x2,y2,x3,y3,x4,y4) {
  // only works on line segments that do intersection &
  // not vertical
  var m12 = (y2-y1)/(x2-x1);
  var m34 = (y4-y3)/(x4-x3);
  var m = m34/m12;
  var x = (x1-y1/m12-m*x3+y3/m12)/(1-m);
  var y = m12*(x-x1)+y1;
  return ([x,y]);
}
```

2.3. Origami Frog

The Japanese word for 'frog' and 'to return' are pronounced the same, \( kaeru \), and so origami and ceramic frogs often were placed on shrines, such as places in the homes of fishermen or other travelers. In any case, there are many origami frog models. This one is a good first lesson, especially because it jumps when you stroke the back as shown in Figure 2.
It also has a certain appeal because business cards are frequently available and often exchanged. The model must be made with a rectangle that is NOT a square and with stiff paper. An index card would also be suitable.

The final (at the time of writing this paper) set of steps for the frog is shown in the declaration statement for the steps array:

```javascript
var steps = [
    [directions, "Diagram conventions"],
    [makebuscard, "Start with a business card..."],
    [adiagonal1p, "...make a diagonal fold."],
    [diagonal1, "Unfold last fold."],
    [adiagonal1, "you see one valley fold."],
    [adiagonal2p, "Now make the opposite diagonal"],
    [diagonal2, "Unfold last fold."],
    [adiagonal2, "Turn over."],
    [turnedover, "You see two mountain folds."],
    [unfoldedp, "Bring top to ends of x to make valley at waist of X."],
    [topdown, "Unfold last fold."],
    [unfolded, "Press sides in to collapse into arrow..."],
    [collapsing, "keep pressing."],
    [arrow, "Turn over."],
    [arrowunder, "Now to make the front legs"],
    [ prerightleg, "Turn up right flap to form leg (this is to taste)."],
    [ preleftleg, "Turn up left flap to match."],
    [ makeleftleg, "Both front legs made."],
    [ prethinned, "Fold in sides to make frog body thinner (to taste)."],
    [ thinned, "Two more folds to go. These are soft folds."],
    [ thinned1, "Fold bottom up to top, soft crease."],
];
```
[foldedup,"Folded up. One more fold."],
[foldback,"Fold bottom back down to fold you just
made, soft crease."],
[doneupsidedown,"Turn over ..."],
[showfrogpicture,"and stroke back to make frog jump!"]
]

Repeat: readers need to understand that I did not write out this list all at once, but
started with the first step, added steps one at a time and tested as I went along.
This included deciding on the utility functions.

As an example, I describe the thought process for calculating the points for the
upturned legs. The table shows the diagrams with critical points labeled. There
actually is not a point mid, but a pair of variables, midx and midy and similarly
for other points.

The challenge here was determining the 'to taste'
for forming the leg. I realized that I could define an
exact position. I decided to calculate the position
1/3 of the way from c to mid. I wrote a function,
proportion, that returned a two-element vector
with the correct x and y values. The line of code
was

```javascript
var rightleg=proportion(cx, cy, midx, midy, .33);
```
The next step was to indicate that the folder is to make a valley fold from the middle of the base of the triangle to the calculated point. The middle of the base is point g and the calculated point to taste is rl. The relevant line of code is:

```javascript
valley(gx,gy,rlx,rly,"green");
```

Now, the challenge is to compute the coordinates for what is labeled `legpoint` in the diagram. The red label `ang` refers to the angle of the smaller triangle. I use the Law of Sines to calculate the smaller angle. The Law of Sines says that the ratio of the sine of each angle in a triangle to its opposite side is the same as the ratio for the other corners. Note that the outer angle for the small triangle (and the whole, 'arrow head' triangle) is 45 degrees, PI/4 in radians, so since the ratio can be computed for this corner and the distance opposite `ang` is known, `ang` can be calculated.

My code doubles the value of `ang` to calculate the green angle and then uses basic trigonometry to compute the `legpoint`. The distance from `g` to `legpoint` is known, being the same as `g` to `c`.

The photograph at the end served the purpose of showing how to make the frog jump. This can be done by a line drawing portraying a three-dimensional scene but I would need to do that plus draw a finger and a photograph seemed much easier and feasible with the HTML5 `drawImage` command.

```javascript
function showfrogpicture() {
    ctx.drawImage(frog,40,40);
}
```
2.4. Origami Fish

The talking fish (aka kissy fish) is an action model made from a square, such as the standard kami paper.

Figure 3 is a screen shot of the video clip shown as the last step of the directions. Video clips are not necessary or even best for much of the folding but for the last step and also an intermediate step involving the sinking of the center square, the video clips serve a critical role.

Video is presented in HTML5 using the video element and, following current practice, each element references three distinct files, each holding the same video content, but made with different encodings. This is to accommodate the lack of standardization on encodings (codecs) for video by the different browsers. For this application, I include two video elements and use a directive in the style element to make them NOT display until the appropriate moment. The relevant coding is in three places. The style element holds

```html
video {display: none;}
```

This makes all video elements not display. The code will change that at the appropriate time.

The body element holds the definitions of the video:
Lastly, the functions to display the steps involving video are similar:

```javascript
function playtalk() {
    v = document.getElementById("talk");
    v.style.display="block";
    v.currentTime = 0;
    v.play();
    canvas1.height = 50;
}

function playsink() {
    v = document.getElementById("sink");
    v.style.display="block";
    v.currentTime = 0;
    v.play();
    canvas1.height = 50;
}
```

Notice that these functions each decrease the height of the canvas element in order to avoid pushing the buttons too far down the screen. The `donext` function erases any video and restores the height of the canvas. Most of the time, these have no effect, but this was the best place to put the code.

```javascript
function donext() {
    if (nextstep>=steps.length) {
```
nextstep=steps.length-1;
}
v.pause();
v.style.display = "none"; //erases last video
canvas1.height = 480; //restore height
ctx.clearRect(0,0,cwidth,cheight);
ctx.lineWidth = origwidth;
steps[nextstep][0]();
ta.innerHTML = steps[nextstep][1];
nextstep++;
}

The fish model inspired me to add the facility of drawing a curved arrow. This was not easy and is the longest function in the application. Arcs in HTML5 are drawn using the center of a circle, the radius, the starting and stopping degrees, and the direction. For this application, I needed multiple cases (vertical, horizontal and other). There also was the challenge of drawing the small lines at the arrow head.

The steps of the directions as captured in the array steps follows. Note: there are no line breaks in the code and each string of text appears on one line.

var steps= [
  [directions,"Diagram conventions"],
  [showkami,"Make quarter turn."],
  [diamond1,"Fold top point to bottom point."],
  [triangleM,"Divide line into thirds and make valley folds and unfold "],
  [thirds,"Fold in half to the left."],
  [rttriangle,"Fold down the right corner to the fold marking a third "]],
  [cornerdow,"Unfold everything."],
  [unfolded,"Prepare to sink middle square by reversing folds as indicated ..."],
  [changedfolds,"note middle square sides all valley folds, some other folds changed. Flip over."],
  [precollapse,"Push sides to sink middle square."],
  [playsink,"Sink square, collapse model."],
  [littleguy,"Now fold back the right flap to center valley fold. You are bisecting the indicated angle."],
  [oneflapup,"Do the same thing to the flap on the left"],
  [bothflapsup,"Make fins by wrapping top of right flap around 1 layer and left around back layer"],
  [finsp,"Now make lips...make preparation folds"],
  [preparelips,"and turn lips inside out. These are reverse folds."],
  [lips,"Pick up fish and look down throat."],
  [showthroat1,"Stick your finger in its mouth and move the inner folded material to one side"],
  [littleguy2,"Now fold back the right flap to center valley fold. You are bisecting the indicated angle."],
  [oneflapup2,"Do the same thing to the flap on the left"],
  [bothflapsup2,"Make fins by wrapping top of right flap around 1 layer and left around back layer"],
  [finsp2,"Now make lips...make preparation folds"],
  [preparelips2,"and turn lips inside out. These are reverse folds."],
  [lips2,"Pick up fish and look down throat."],
  [showthroat2,"Stick your finger in its mouth and move the inner folded material to one side"],
];
The addition of the `showthroat` functions was a late one. I decided it was helpful to the viewer to see a photograph of the fish throat.

To illustrate the use of the coordinate system transformation functions, here is the code for producing the last line drawing:

```javascript
function rotatefish() {
    ctx.save();
    ctx.translate(kx,my);
    ctx.rotate(-Math.PI/2);
    ctx.translate(-kx,-my);
    lips();
    ctx.restore();
}
```

This function saves and then restores the original coordinate system. You can say that since this is the last drawing, it isn't necessary, but it is a good practice. The first translate command moves the coordinate system to a point on the model. The rotate command rotates 90 degrees. The second translate command moves the coordinate system to a position analogous to using the next function to draw the whole model.

### 2.5. Porting to Processing

Processing is a language developed by visual artists, with and “on” Java. Processing programs are called sketches. It comes with its own Development Environment. The code is essentially Java code and a processing sketch can be ported to a Java IDE with minor adjustments and with the addition of the Processing libraries. Processing is a compiled, strongly-typed language. Processing is Open Source and there are many books and websites available.

A benefit of using HTML5 and JavaScript is that work can be shared by uploading to a website. Moreover, the code can be examined using the View Source standard feature in any browser. Sharing work is more of a challenge in Processing. The community does have an effort called Processing JS with which many Processing sketches can be disseminated on the web. However, at this time, video is problematic in Processing JS.
The conversion of a HTML5 and JavaScript document to a Processing sketch was, as I have noted, relatively easy. I needed to add variable declarations to specify datatypes and also modify function headers, specifically including datatypes of parameters and return values. Displaying an image and starting and stopping a video clip is possible in Processing.

A more troublesome requirement was that functions are not regular values and so I could not use the steps array to list the combined function name and the direction string. Instead, I coded a case statement in which the number corresponding to the step triggered a call to the corresponding function. I would like an alternative solution.

I note that Processing has some support for 3D. It is not a fully-functional 3D modeling system, more just 3D sketching. However, it may provide useful features for origami, which I may explore in the future.

3 Discussion

Though the code may appear daunting, the applications came together relatively quickly, only a few hours for each one. I re-wrote each when I finally came to terms with how to show unfolded folds. I added the picture to the frog and the two video clips and then the two pictures to the fish later. Lastly, I added the very first step for showing the conventions for lines. These changes were easy to do because of the flexibility of the approach, including the structure of the steps array.

My original objective was to demonstrate HTML5 JavaScript capabilities and I consider it a success. Origami directions provided a justified use of a mixture of line drawings, photographs, and video clips. However, the calculation of position points using mathematics also demonstrated the power of computer software to solve actual problems. It may be intriguing to teachers and students of algebra, geometry and trigonometry. Lastly, beginners at programming often balk at defining functions, thinking it makes their programming task more difficult, not easier. The origami examples showcase how programming involves the definition of general, well-defined functions that may be invoked multiple places in the application. The general function may be easier than the special-case. I am confidant that other origami models will produce similar experiences in mathematics and programming while demonstrating the magic of origami.

References


Processing, https://processing.org/.