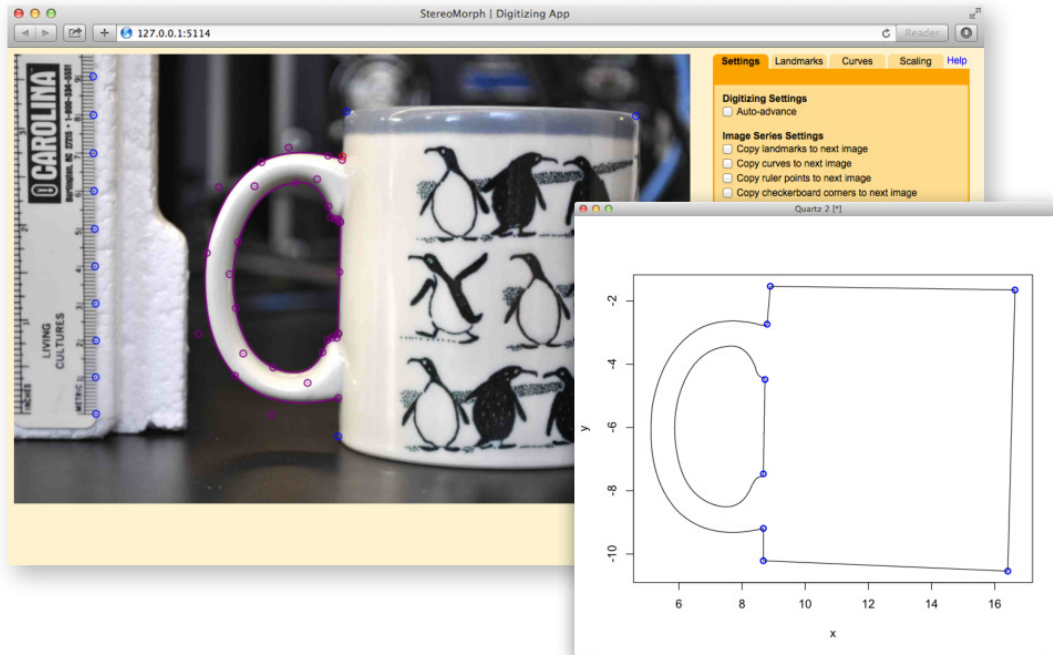


Digitizing with StereoMorph

How to collect 2D landmark and curve data from photographs using the StereoMorph Digitizing App



Aaron Olsen
April 6, 2015

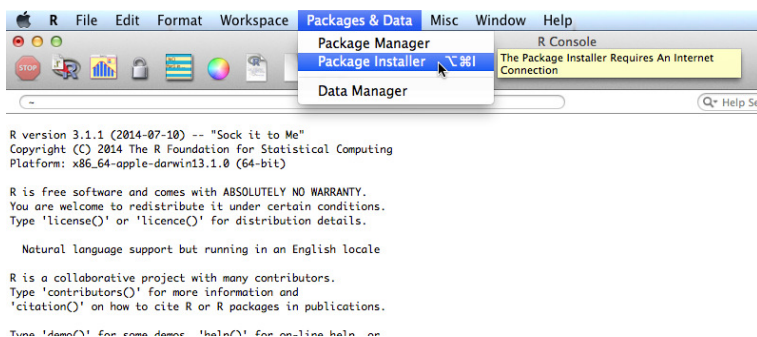
Table of Contents

1	Getting started	3
2	Launching the app	5
3	Digitizing landmarks	8
4	Digitizing curves	11
5	Scaling landmarks and curves	15
5.1	Scaling with a ruler	15
5.2	Automated scaling with a checkerboard	19
6	Reading saved shape data	23
7	Sub-sampling curve points	27
8	Shortcuts (keyboard and otherwise)	29
9	Where to go for help	30
10	Citing StereoMorph	30

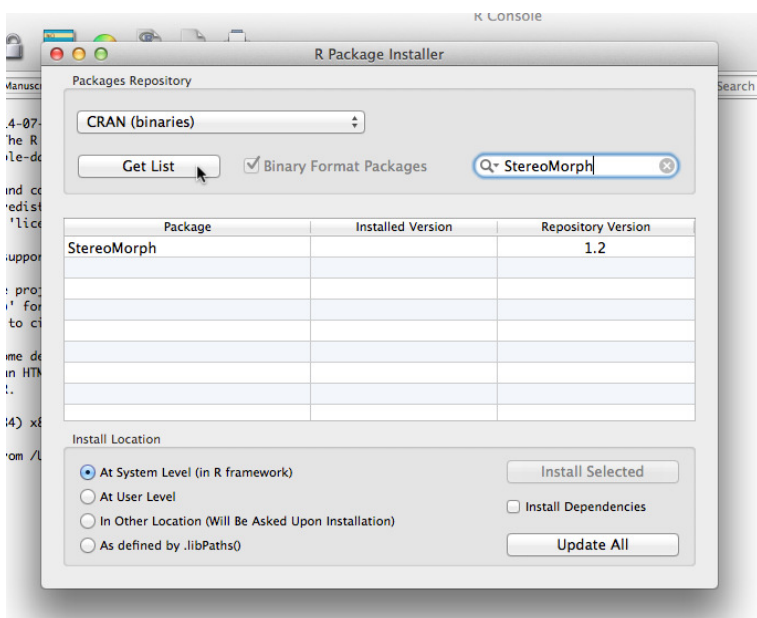
1 Getting started

This tutorial will show you how to collect 2D landmark and curve data from photographs using the R package [StereoMorph](#). The R project is a computing language and platform that allows users to freely upload and share software packages. The StereoMorph package includes a digitizing app that runs in a web browser, providing an interactive and easy-to-use interface. The app is also completely free and compatible across Linux, Mac and Windows. The following steps will take you through process of installing the StereoMorph package.

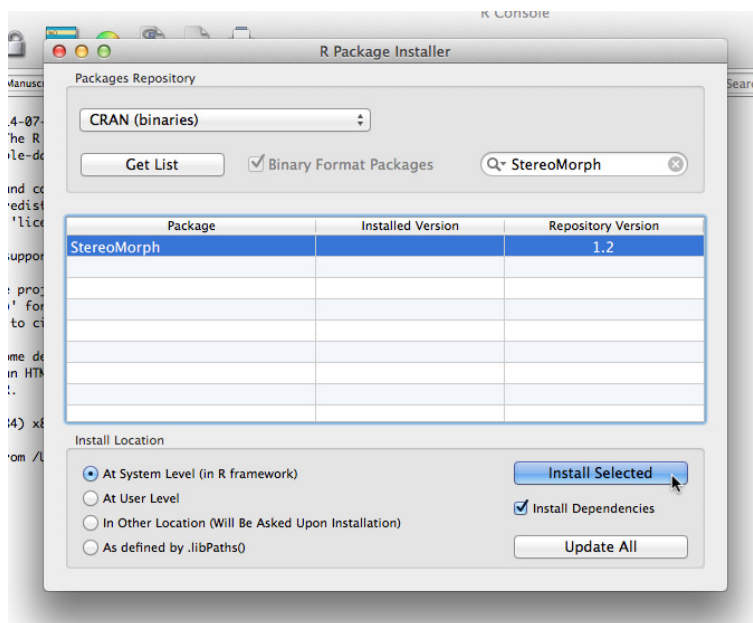
1. If you do not already have R installed on your computer, begin by [installing R](#). R can be installed on Windows, Linux and Mac OS X.
2. Once installed, open R.
3. Go to Packages & Data > Package Installer.



4. Find the StereoMorph package binary by typing “StereoMorph” into the Package Search box and clicking Get List. (The repository version of StereoMorph will be greater than the version in the image below).



5. Check the box next to Install Dependencies. This ensures that all the packages that StereoMorph requires to run will be installed as well. Then click Install Selected to install StereoMorph.



6. Load the StereoMorph package into the current R session using the library command.

```
> library(StereoMorph)
```

For the rest of this tutorial, text in typewriter font preceded by a “>” will be used to indicate commands to be entered into the R console.

7. If you would like to download the example images and files used in this tutorial you can download the folder [here](#) (300 KB). Unzip the folder anywhere you’d like on your system.

8. Lastly, you’ll need a compatible web browser to use the digitizing app. The app is currently compatible with Safari, Chrome and Opera.

You are now ready to launch the digitizing app!

2 Launching the app

Once you've installed R and the StereoMorph package (see [Getting started](#)), you're ready to launch the digitizing app. The app is launched in R but runs in the user's default web browser so you'll need to make sure one of the three compatible web browsers (Safari, Chrome and Opera) is your default web browser. And although the app runs in a browser you do not have to be connected to the internet to use it (the app runs on a local server).

1. Open R and load the StereoMorph library (see [Getting started](#)).

```
> library(StereoMorph)
```

2. It is easiest to set the working directory in R to be the folder containing the image or folder of images you want to digitize. You can do this by going to “Misc & Data” > “Change Working Directory...” or using the `setwd()` function. If you are working through the [tutorial project](#), you'll set the working directory to the “Mug Project” folder. This contains a folder, “Mugs”, that has three images of coffee mugs. This is what the `setwd()` looks like on my system:

```
> setwd('/Users/aaron/Documents/Mug project')
```

You can copy-and-paste this in or, on some operating systems, you can simply drag the folder into the R console and the path will appear.

3. Launch the app using the function `digitizeImage()`. For the tutorial example project, the function call looks like this:

```
> digitizeImage(image.file='Mugs', shapes.file='Shapes',  
               landmarks.ref='landmarks.txt')
```

Now to explain each of these inputs.

image.file This argument specifies the path to the images that you are going to digitize. This can either be a path to a particular image or a path to a folder containing several images. The digitizing app supports three image file types: tiff, jpeg and png. If “image.file” is a folder containing several images you will be able to click through the images within the app. For the tutorial example we use “Mugs”, since this is the folder that contains the photos of the coffee mugs.

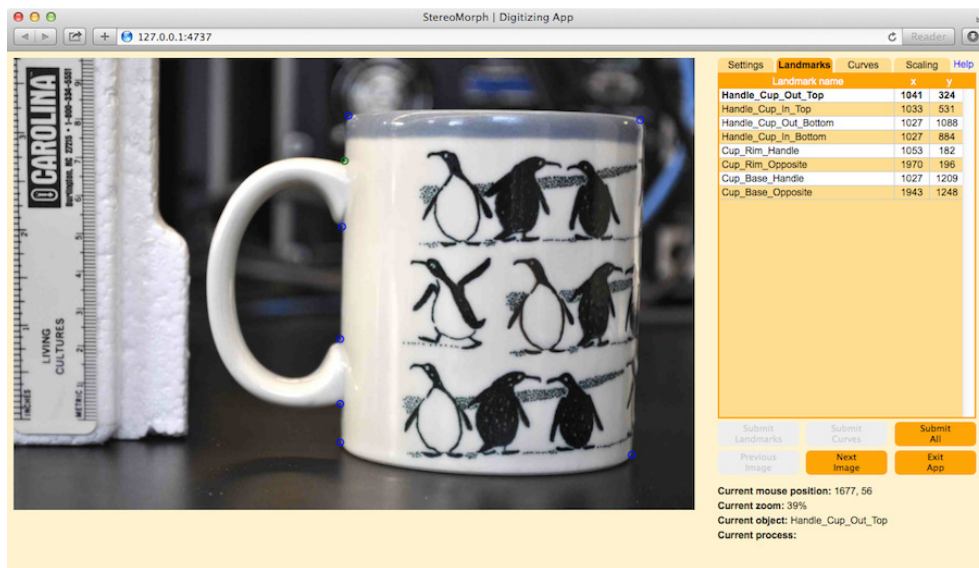
shapes.file This argument specifies where to save the shape data that you collect. The shape data file will be saved as a text file in a format that can be read by StereoMorph. Similar to “image.file”, this can either be a path to a particular file or, as here, a path to a folder (“Shapes”) where the shape data files are to be saved. The “image.file” and “shapes.file” arguments should be of the same type (i.e. both folder or both a single file). For the tutorial example these will be saved to the folder “Shapes”.

landmarks.ref This argument specifies the names of the landmarks to be collected. This can either be a vector of names or a path to a text file containing a list of the landmark names. The file “landmarks.txt” in the tutorial folder contains a list of landmarks for digitizing the coffee mugs.

If you wanted to launch the app with a single image instead, you would replace the folder names with filenames:

```
> digitizeImage(image.file='Mugs/mug_001.jpg',
  shapes.file='Shapes/mug_001.txt', landmarks.ref='landmarks.txt')
```

Once you call `digitizeImage()`, the digitizing app will launch with the first image.



The StereoMorph digitizing app after loading first image.

The left two-thirds of the window are the image frame. This is where you can navigate around the image and add landmarks and curves using the mouse.

4. Navigate around the image in the image frame using the same basic mouse actions as in Google Maps. Position your cursor somewhere over the image and scroll just as you would scroll up and down a web page. This will zoom in and out of the image. To more quickly navigate around the image, the zoom tracks the position of your cursor and zooms in and out of particular region of the image based on the current position of your cursor. For instance if you wanted to zoom in to the bottom-left corner of the image, you would position your cursor in the bottom-left of the image and scroll in the appropriate direction. This will increase the size of the image while simultaneously positioning the bottom-left corner of the image into the center of the image frame.



Zoom into the image for precise landmark positioning.

5. Once you've zoomed in, click and drag somewhere on the image. This will cause the image to move with your cursor.

To the right of the image frame is the control panel. This contains all of the tools for collecting and saving shape data and will be covered in the following sections.

3 Digitizing landmarks

Landmarks are features on an object that can be described by or localized to a single point. When digitizing landmarks from a 2D image you should only use points that are approximately in the same plane and this plane should be parallel to the end of the camera lens. This is because of the perspective effect: distances measured closer to the camera will appear larger than the same distance measured further away. On a coffee mug you could define eight landmarks based on the “corners” of the cup and where the handle attaches to the cup. Of course we’re missing a lot of information on the three-dimensional shape of the mug. But all of these landmarks are more or less in the same plane and provide some basic information on the mug’s shape.

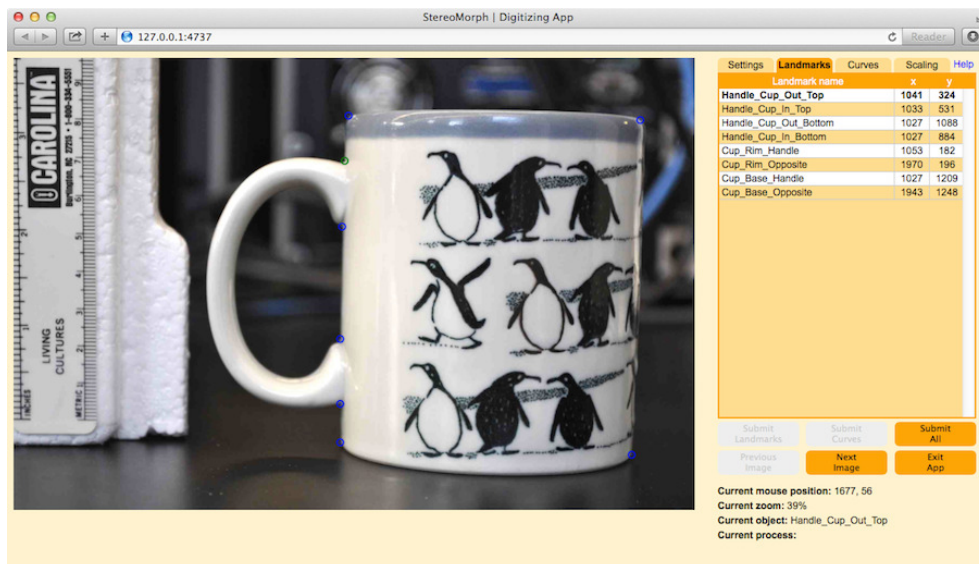


Eight landmarks (in red) digitized on a coffee mug.

1. Before you start digitizing landmarks, you’ll need to specify the names of the landmarks you want to collect. You can specify these as a vector in the R console but it is easiest to do this in a text file, especially for a long list of landmarks. In the [example tutorial folder](#) you’ll find a file named “landmarks.txt” with a list of the coffee mug landmarks. List each landmark name on a separate line of the file. The landmark names can be anything you’d like, with letters or numbers, but should not contain spaces (use underscores instead).
2. Launch the digitizing app in the R console using `digitizeImage()` (to run the tutorial code, make sure you’ve set the working directory to the “Mugs Project” folder, see [Launching the app](#)):

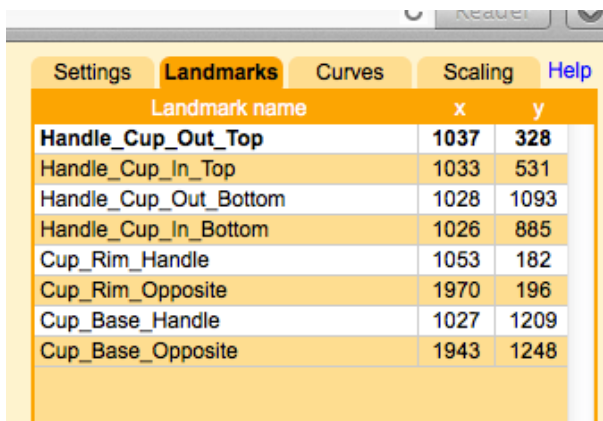
```
> digitizeImage(image.file='Mugs', shapes.file='Shapes',  
  landmarks.ref='landmarks.txt')
```

When the app opens you’ll see digitized landmarks on the mug in the image frame.



The launched app with digitized landmarks (in blue).

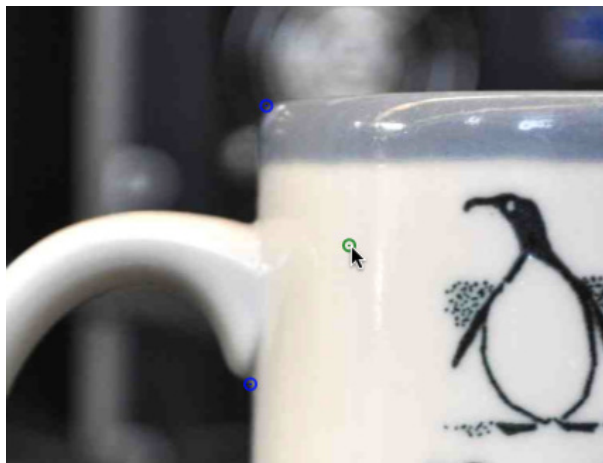
- Click on the “Landmarks” tab in the control panel at the right to see the list of landmarks by name and their corresponding pixel coordinates. You’ll see that the first landmark is bold by default.



The first landmark selected in the Landmarks panel.

In the image frame this landmark is also green while the rest are blue. This means that it is currently selected.

- Move this landmark by clicking and dragging the green landmark in the image frame to a new location. Try zooming into the selected landmark to position it more precisely.



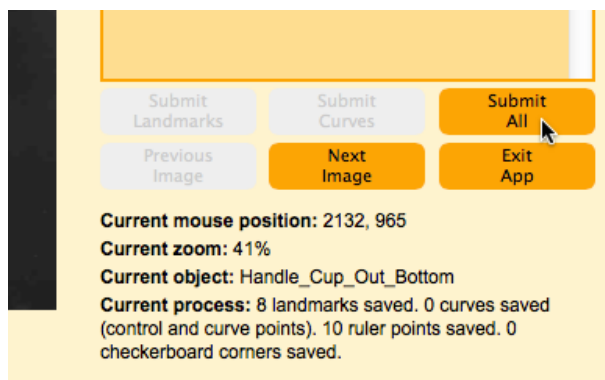
Move landmarks by clicking and dragging with the mouse.

You can also move landmarks using the arrows on the keyboard. This will move the landmarks in single pixel increments (hold shift to move in 10 pixel increments).

5. To select a different landmark, you can either click on another landmark in the Landmarks panel or you can double-click on the landmark in the image frame (the keyboard equivalent to the double-click is the letter “x”). You can also use the “n” and “p” keys to select the next or previous landmark, respectively.

6. To delete a landmark first select the landmark and then press “d” on the keyboard. The pixel coordinates of the landmark in the Landmarks panel will change to “-”. Note that while the landmark name will still appear in the Landmarks panel, any landmarks with a position “-” will not be saved. To choose a new position for the landmark first make sure the landmark is selected in the panel and then double-click over its new position in the image frame (or press the “x” key).

7. To save the landmarks, click the “Submit All” button in the bottom right corner of the window.



Save landmarks by clicking “Submit All”.

A message will appear next to “Current process” telling you the shape data that have been saved.

4 Digitizing curves

The StereoMorph digitizing app also includes tools for digitizing curves, such as for describing the shape of the coffee mug handle. Specifically, the app uses Bézier curves joined end-to-end (called Bézier splines) that can fit pretty much any natural curve.



Curves (in green) describing the coffee mug handle.

The app assumes that each curve you digitize is bound at either end by landmarks. You don't have to create a list of landmarks to digitize curves but if you don't specify any the app will automatically add them to correspond to the start and end points of the curve.

1. Start by specifying the names of the curves you want to collect, including the start and end landmarks. It is easiest to define these in a separate text file. On each line, list the name of the curve, the starting landmark and the ending landmark separated by tabs. In the [example tutorial folder](#) you'll find a file named "curves.txt" with a list of the coffee mug curves. The first three lines of the file look like this:

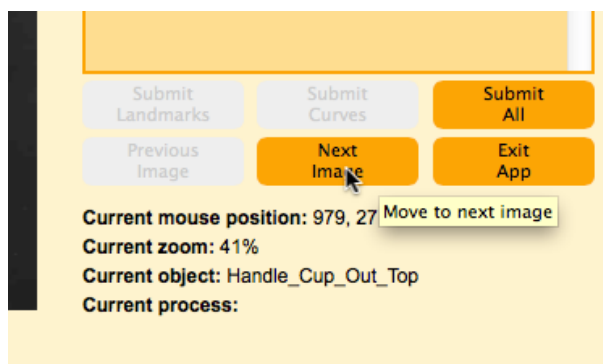
```
Handle_Cup_Out      Handle_Cup_Out_Top   Handle_Cup_Out_Bottom
Handle_Cup_In       Handle_Cup_In_Top    Handle_Cup_In_Bottom
Side_Handle_Rim     Cup_Rim_Handle       Handle_Cup_Out_Top
```

The curve names can be anything you'd like, with letters or numbers, but should not contain spaces (use underscores instead). Also the curve names (first column) should not be the same as any of the landmark names.

2. Launch the digitizing app in the R console using `digitizeImage()` (to run the tutorial code, make sure you've set the working directory to the "Mugs Project" folder, see [Launching the app](#)):

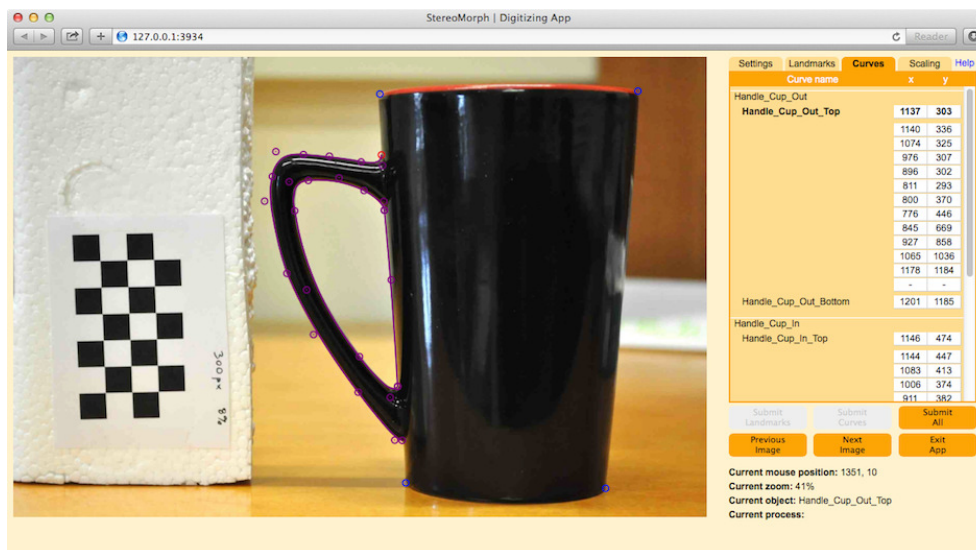
```
> digitizeImage(image.file='Mugs', shapes.file='Shapes',
  landmarks.ref='landmarks.txt', curves.ref='curves.txt')
```

3. Once the app opens click on "Next Image" in the bottom right of the window to advance to the next coffee mug image.



Advance to the next image by clicking "Next Image".

You'll see the second coffee mug image with digitized landmarks and curves.



The second mug with digitized landmarks (blue) and curves (purple).

4. Click on the "Curves" tab in the control panel at the right to see the list of curves by name and their corresponding pixel coordinates.

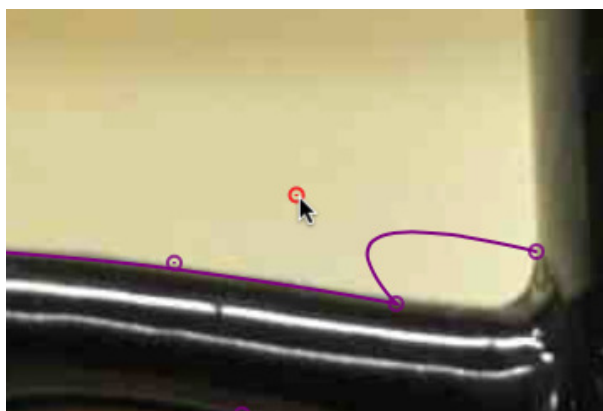
Curve name	x	y
Handle_Cup_Out		
Handle_Cup_Out_Top	1137	303
	1140	336
	1074	325
	976	307
	896	302
	811	293
	800	370
	776	446
	845	669
	927	858
	1065	1036
	1178	1184
	-	-
Handle_Cup_Out_Bottom	1201	1185
Handle_Cup_In		
Handle_Cup_In_Top	1146	474
	1144	447

The curves digitized on the second mug.

There's a subsection in the Curves panel for each curve with the curve name, the start and end landmarks and several points in between that define the shape of the curve. The first point, "Handle_Cup_Out_Top", in the first curve, "Handle_Cup_Out", should be selected and the corresponding point in the image frame should be red.

5. Move the selected point by clicking and dragging it in the image frame. As you move the point, the curve will be re-drawn.

6. In the Curves panel, select the second point from the top and zoom into this point in the image frame by scrolling. This point does not have a name or correspond to any landmarks. Also, the Bézier curve does not pass through this point but instead "reaches" out to the point. Click-and-drag this point to re-shape the curve.



Move the control points to define the Bézier curve.

7. Select the next point (the third from the top) either by clicking it in the control panel or pressing the "n" key. Notice that the curve does pass through this point. As you click

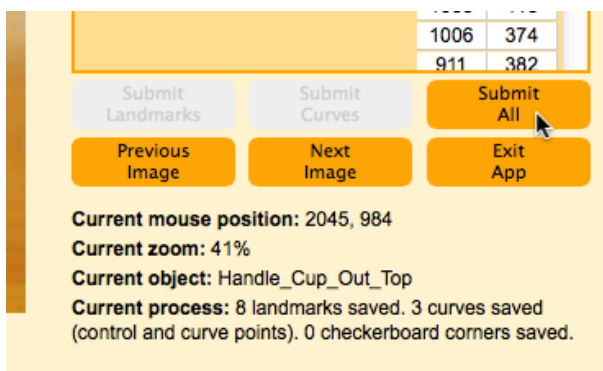
through the curve points you'll see they alternate between points that the curve passes through and points that the curve reaches toward. Each anchor point the curve passes through forms a Bézier curve and these are all joined end to end to form a Bézier spline.

8. Click on the second to last field of the “Handle_Cup_Out” curve (the one with the “-”’s) and then click anywhere in the image. This will add a control point, breaking the curve.

9. Click on the next field filled with “-”’s (or press “n”) and then click somewhere else. This will complete the curve again. You can continue adding points and to form the curve with as many component Bézier curves as you need. Just note that you'll always have to add two points at a time so that the entire curve is continuous.

10. You can delete any of the curve points by selecting the point and pressing “d”. But again you'll have to delete pairs of points to keep the curve continuous.

11. To save the curves, click the “Submit All” button in the bottom right corner of the window.



Save curves by clicking “Submit All”.

A message will appear next to “Current process” telling you the shape data that have been saved. For curves, two sets of data are saved. The first are the control points that you see in the Curves panel. The app also saves all of the points along the curve at single pixel spacing.

5 Scaling landmarks and curves

The app allows users to scale landmarks and curves using two methods. The first is to digitize two or more points along a ruler and the second is to include a checkerboard pattern in the image and have the app automatically detect the size of the checkerboard squares. The user then specifies either the distance between the ruler points or the size of the checkerboard squares in “real-world” units (e.g. cm) and the app calculates the corresponding scaling.

5.1 Scaling with a ruler

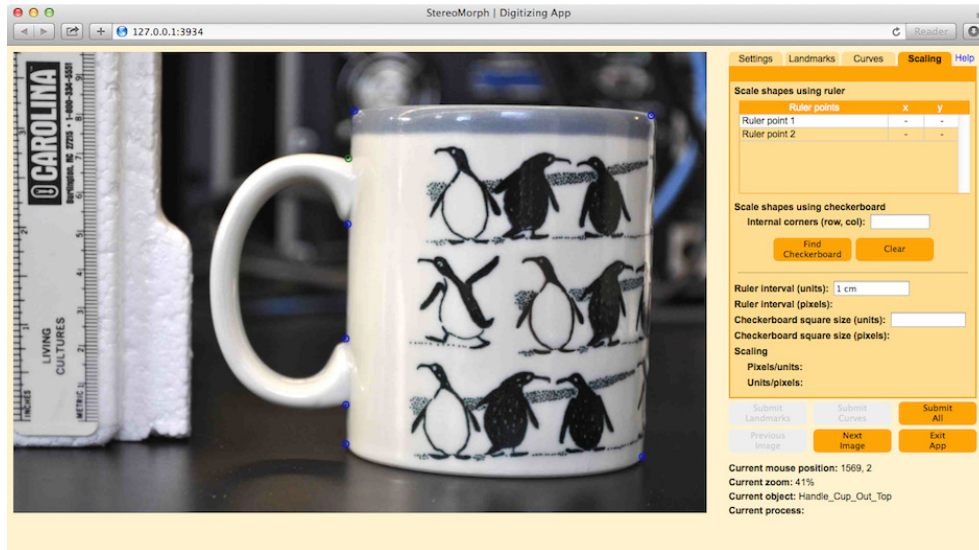
1. To scale shape data using a ruler first make sure that in your images you have a ruler that is positioned more or less in the same plane as the landmarks or curves you are digitizing.



Coffee mug with digitized landmarks and a ruler.

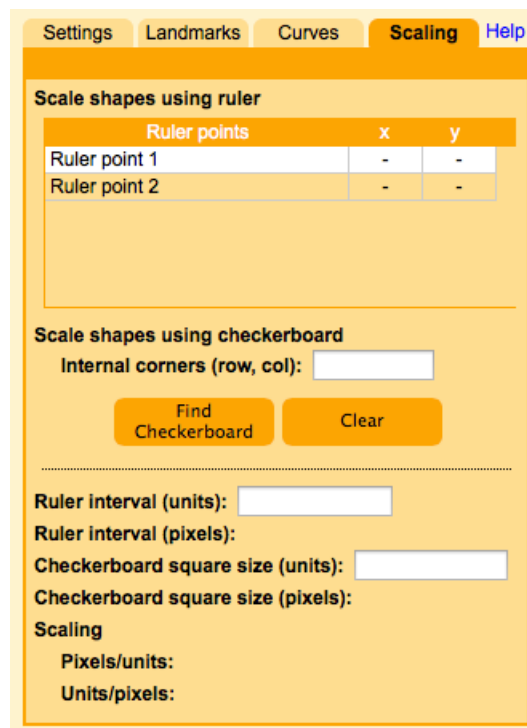
2. Launch the digitizing app in the R console using `digitizeImage()` (to run the tutorial code, make sure you've set the working directory to the “Mugs Project” folder, see [Launching the app](#)):

```
> digitizeImage(image.file='Mugs', shapes.file='Shapes',  
                landmarks.ref='landmarks.txt')
```



Scaling shape data using a ruler.

4. In the control panel, click on the “Scaling” tab. At the top you’ll see “Scaling shapes using ruler” and just below it is a table for ruler points similar to the table for landmarks.



The scaling panel.

5. Click on “Ruler point 1” to select the first ruler point.

6. Then, zoom into the ruler on the left side of the image by scrolling and add the first ruler point on the zero line of the metric ruler by double-clicking or pressing “x”.



Adding the first ruler point.

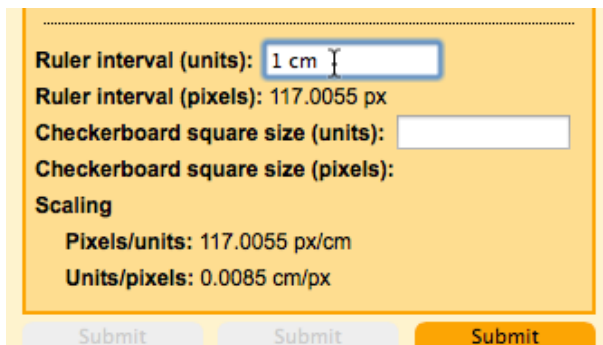
7. Select “Ruler point 2” and add a second ruler point at the first centimeter mark. If you look at the lower half of the Scaling panel you’ll notice that a value has appeared next to “Ruler interval (pixels)”, it should be about 117 pixels.

The estimated ruler interval in pixels.

This is the distance between the two digitized ruler points.

8. Select “Ruler point 3” (or press “n” and add a third ruler point at the next centimeter mark. The “Ruler interval (pixels)” field will automatically update the length. If more than two ruler points are selected the app finds the distance between the points by model fitting; this is different than a simple average and results in a less biased estimated of the interval. You can continue to add as many ruler points as you’d like and the app will continue to update the estimated interval. Two ruler points should be sufficiently accurate for most applications but in that case it’s best to position the points at opposite ends of the ruler.

9. This gives us the ruler interval in pixels but we want a scaling factor from pixels to “real-world” units. To get this, enter the length of the ruler interval including units (ex. cm, inches) in the “Ruler interval (units)” field.



The screenshot shows a yellow-themed control panel with the following text and input fields:

- Ruler interval (units):** 1 cm (with a cursor in the input field)
- Ruler interval (pixels):** 117.0055 px
- Checkerboard square size (units):** [empty input field]
- Checkerboard square size (pixels):** [empty input field]
- Scaling**
 - Pixels/units:** 117.0055 px/cm
 - Units/pixels:** 0.0085 cm/px

At the bottom, there are three buttons labeled "Submit". The rightmost button is highlighted in orange, while the other two are grey.

Entering the ruler interval in units.

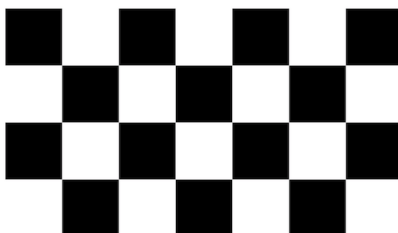
Once you enter a number in this field the app will automatically calculate the appropriate scaling factor.

10. Click the “Submit All” button to submit the shape and scaling data. The ruler points and scaling factors will be saved along with the shapes so they’ll automatically appear if you re-open the same image and shape file. Also, any landmark and curve data will be saved both as pixel coordinates and with the scaling factor applied.

5.2 Automated scaling with a checkerboard

An alternative to using a ruler to scale shape data is to use a checkerboard of a known square size. The StereoMorph package can automatically detect the internal corners of a checkerboard which can then be used to estimate the square size, eliminating the need to manually digitize points on a ruler. The automated checkerboard detection currently only works with jpeg images.

1. Start by printing a checkerboard onto a piece of paper. Card stock paper works best since it stays flat but you can also tape regular printer paper to a hard, flat surface. The [example tutorial folder](#) contains a 7x4 square checkerboard.



Checkerboard with 7x4 squares and 6x3 internal corners.

You can learn how to create a checkerboard of different dimensions [here](#).

2. Once you print out the checkerboard you'll need to measure the size of the squares. You can learn [here](#) how to measure the checkerboard square size.
3. In the image make sure that the checkerboard is positioned more or less in the same plane as the landmarks or curves you are digitizing.

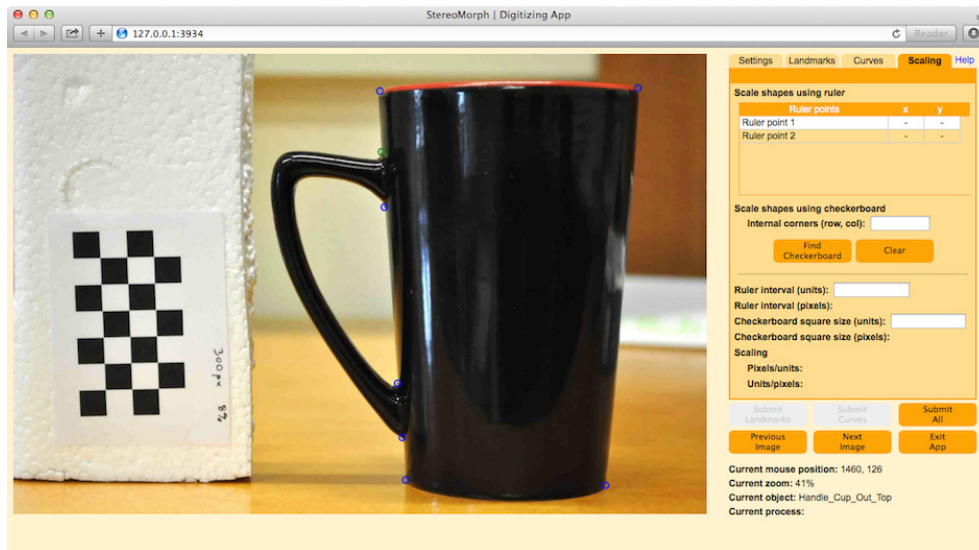


Coffee mug with digitized landmarks and a checkerboard.

4. Launch the digitizing app in the R console using `digitizeImage()` and then click "Next Image". This will advance to the second coffee mug image, which also has a checkerboard.

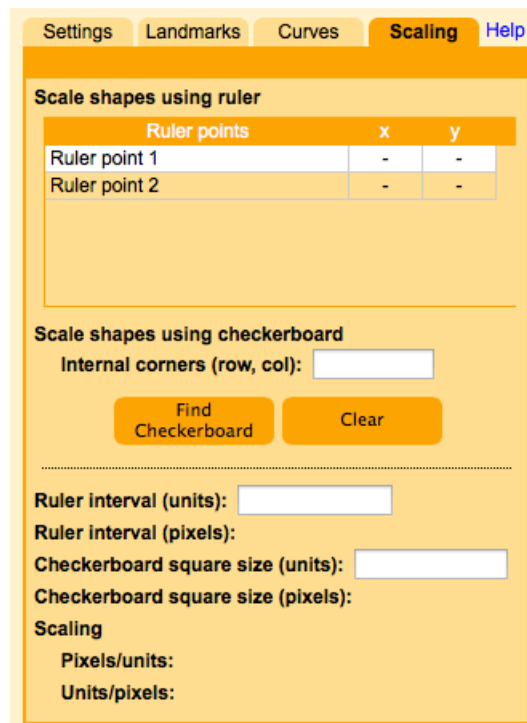
```
> digitizeImage(image.file='Mugs', shapes.file='Shapes',
  landmarks.ref='landmarks.txt')
```

To run the tutorial code, make sure you've set the working directory to the “Mugs Project” folder (see [Launching the app](#))



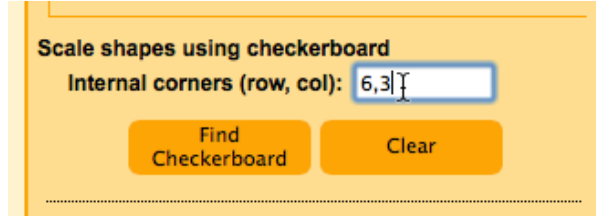
Scaling shape data using a checkerboard.

5. In the control panel, click on the “Scaling” tab. In the middle of the panel you’ll see “Scaling shapes using checkerboard”.



The scaling panel.

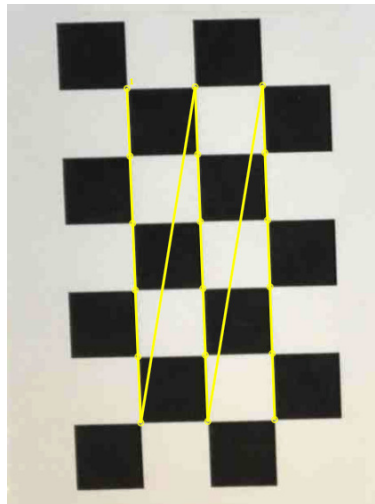
6. In the “Internal corners” text field below this enter the number of internal corners along each dimension. These are the points where the corners of the black squares meet. For the tutorial images, the checkerboard has 6 internal corners along one dimension and 3 along the other.



Entering the internal corners of the checkerboard.

The order in which you list these does not matter, so you could also enter “3,6”.

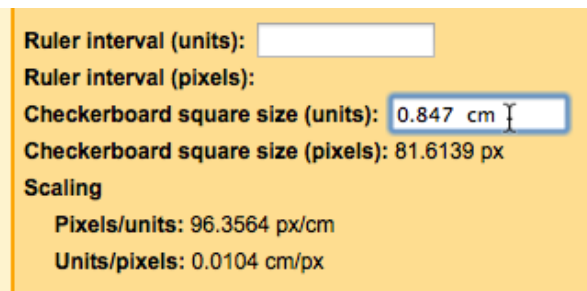
7. Click “Find Checkerboard”. The app will then try to automatically detect the checkerboard pattern. As long as the entire checkerboard is clearly visible and the lighting is uniform across the pattern, the app should succeed in finding the pattern. Once the corners are detected, they’ll be displayed on the image as a series of yellow points joined by yellow lines.



Automatically detected internal checkerboard corners.

The app will also calculate the square size of the checkerboard in pixels and print this next to “Checkerboard square size”.

8. This gives us the square size in pixels but we want a scaling factor from pixels to “real-world” units. To get this, enter the size of the checkerboard square including units (ex. cm, inches) in the “Checkerboard square size (units)” field.



The screenshot shows a yellow panel with the following text and input fields:

- Ruler interval (units):**
- Ruler interval (pixels):**
- Checkerboard square size (units):**
- Checkerboard square size (pixels):** 81.6139 px
- Scaling**
 - Pixels/units:** 96.3564 px/cm
 - Units/pixels:** 0.0104 cm/px

Entering the checkerboard square size in units.

Once you enter a number in this field the app will automatically calculate the appropriate scaling factor.

9. Click the “Submit All” button to submit the shape and scaling data. The checkerboard points and scaling factors will be saved along with the shapes so they’ll automatically appear if you re-open the same image and shape file. Also, any landmark and curve data will be saved both as pixel coordinates and with the scaling factor applied.

6 Reading saved shape data

Once you’ve digitized and saved shape data you’ll want to read these data from the files for plotting, running analyses, etc. The shape files are saved in a format that can be read by the StereoMorph function `readShapes()`. You can easily obtain the landmarks and curves as matrices or arrays from `readShapes()` in the R console and then save these data structures to files or run analyses on them directly.

1. Open R and load the StereoMorph library, if it’s not already loaded (see [Launching the app](#)).

```
> library(StereoMorph)
```

2. Call the `readShapes()` function with the file path of the shapes file you want to read. For the [Mugs Project tutorial](#), the shape files are saved in the “Shapes” folder. We’ll start by reading in the first shape file, “mug_001.txt”, and save the output to “read_shapes”.

```
> read_shapes <- readShapes('Shapes/mug_001.txt')
```

3. Type “read_shapes” to print the content in the console.

```
> read_shapes
```

The output is formatted so that it’s easier to read all of the contents of the object. The object names are listed to the left. To get a particular object, add “\$” and the object name after `read_shapes`. For example, to get a matrix of the landmarks in pixel coordinates, use “landmarks.pixel”:

```
> read_shapes$landmarks.pixel
      [,1] [,2]
Handle_Cup_Out_Top    1041  324
Handle_Cup_In_Top    1033  531
Handle_Cup_Out_Bottom 1027 1088
Handle_Cup_In_Bottom 1027  884
Cup_Rim_Handle        1053  182
Cup_Rim_Opposite      1970  196
Cup_Base_Handle       1027 1209
Cup_Base_Opposite     1943 1248
```

If you digitized ruler points in the first image and entered a ruler interval (see [Scaling landmarks and curves](#)) to calculate a scaling factor you can get that factor here using the following:

```
> read_shapes$scaling
[1] 0.008445645
```

You can also get the scaled landmarks using “landmarks.scaled”:

```
> read_shapes$landmarks.scaled
              [,1]      [,2]
Handle_Cup_Out_Top      8.791916 -2.736389
Handle_Cup_In_Top       8.724351 -4.484637
Handle_Cup_Out_Bottom   8.673677 -9.188861
Handle_Cup_In_Bottom    8.673677 -7.465950
Cup_Rim_Handle          8.893264 -1.537107
Cup_Rim_Opposite       16.637920 -1.655346
Cup_Base_Handle        8.673677 -10.210784
Cup_Base_Opposite     16.409888 -10.540165
```

Note that the second column of the scaled landmarks (the y-coordinates) are negative. This is so that when they are plotted the orientation matches the orientation in the image. In pixel coordinates, it is customary to make the top-left corner of the image $\{0,0\}$ and increase the y-values positively as you move down the image (you'll see this as you move the cursor around the image in the digitizing app). As a result, when the pixel coordinates are plotted in the traditional Cartesian system, they will appear upside-down relative to their orientation in the image. Flipping the y-values corrects for this.

4. In order to read shape data from multiple files at once, call `readShapes()` on the folder containing all of the shape files. This is the “Shapes” folder in the tutorial.

```
> read_shapes <- readShapes('Shapes')
```

5. Use “`read_shapes$landmarks.pixel`” to get the landmarks in pixel coordinates from all of the shape files.

```
> read_shapes$landmarks.pixel
```

Now, instead of a single landmark matrix, the landmarks are stored in an array (a set of matrices). Since there are three images, the array contains three matrices, each corresponding to the landmarks from a different image. To get only the landmarks from the first image, use “`mug_001`” as below.

```
> read_shapes$landmarks.pixel[, , 'mug_001']
```

If the number or names of the landmarks differ between images, `readShapes()` will automatically add these to the entire array. Any landmarks that are missing from an image will have values of “NA”.

6. Curve points can be obtained in similar way. First, use the `readShapes()` function to read in the shape data from the second coffee mug.

```
> read_shapes <- readShapes('Shapes/mug_002.txt')
```

7. Use the following to get the curve points for the “Handle_Cup_Out” curve in pixel coordinates.

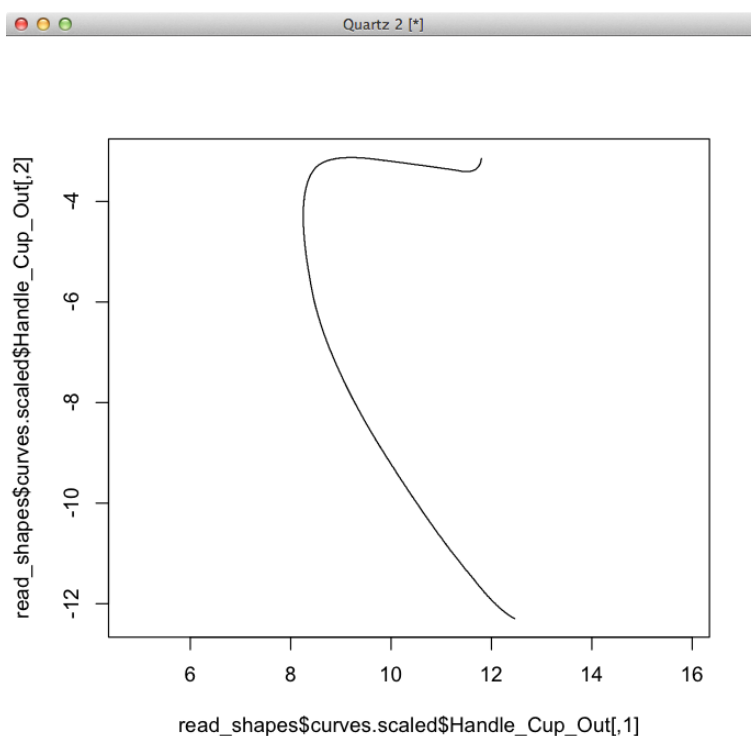

```
> read_shapes$curves.pixel$Handle_Cup_Out
```

This is a large matrix (about 1200 rows) since the path of the curve is described by adjoining pixels (points at pixel resolution). If you used the automated checkerboard finding and scaled the curves in the second image (see [Scaling landmarks and curves](#)), you can get the scaled curve points using “curves.scaled”.

```
> read_shapes$curves.scaled$Handle_Cup_Out
```

8. Plot the curve points using the plot function.

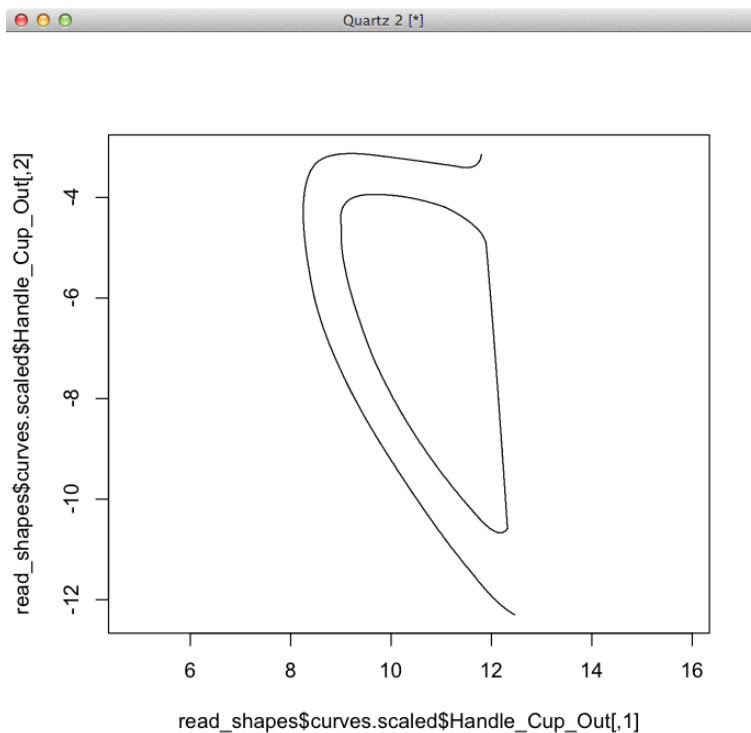
```
> plot(read_shapes$curves.scaled$Handle_Cup_Out, asp=1, type='l')
```



Plotting the outer curve of the coffee mug handle.

9. While the plot window is still open, add the remaining curves to the plot

```
> points(read_shapes$curves.scaled$Handle_Cup_In, type='l')
> points(read_shapes$curves.scaled$Side_Handle_In, type='l')
```



Plotting the curves of the coffee mug handle.

10. To retrieve curve data from multiple images, call the `readShapes()` function with a folder containing multiple shape files or a vector of shape files.

```
> read_shapes <- readShapes('Shapes')
```

11. The curves from multiple shape files can't be placed into the same array as with landmarks. Instead, you have to specify which image you want to retrieve curves from. For example, to get scaled curve points for the outer cup handle from the last coffee mug, you would use the following:

```
> read_shapes$curves.scaled$mug_003$Handle_Cup_Out
```

7 Sub-sampling curve points

The curve points that are returned from the digitizing application are at the resolution of the image pixels. This includes both “curves.pixel” (at pixel scaling) and “curves.scaled” (scaled using ruler or checkerboard points). This means that a homologous curve digitized across different photographs will have different numbers of points. This also means there will be far more curve points than you would like to use in shape analyses. For most curves only a small subset of points are necessary to describe most of the curve’s features.

To fix both of these issues we can sub-sample the points on the curve. The StereoMorph function `pointsAtEvenSpacing()` will return a specified number of evenly spaced points along a curve. Note that this “even spacing” is measured along the curve (i.e. arc length) not between the points themselves. So the straight-line distances between the returned points will not necessarily be the same between pairs of adjacent points.

1. Open R and load the StereoMorph library, if it’s not already loaded (see [Launching the app](#)).

```
> library(StereoMorph)
```

2. Use the `readShapes()` function to read in the digitized shape data for “mug_001”. For the [Mugs Project tutorial](#), the shape files are saved in the “Shapes” folder. From these shapes we will just take the curves scaled to centimeters and save this list of 2D matrices to “curves”.

```
> curves <- readShapes('Shapes/mug_001.txt')$curves.scaled
```

3. To sub-sample different numbers of points for each curve start by creating a new list for the sub-sampled curve points.

```
> curves_sub <- list()
```

4. Set the number of points you’d like to sub-sample from each curve using the “n” parameter.

```
> curves_sub[['Handle_Cup_Out']] <- pointsAtEvenSpacing(
  x=curves[['Handle_Cup_Out']], n=15)
> curves_sub[['Side_Handle_In']] <- pointsAtEvenSpacing(
  x=curves[['Side_Handle_In']], n=5)
> curves_sub[['Handle_Cup_In']] <- pointsAtEvenSpacing(
  x=curves[['Handle_Cup_In']], n=10)
```

4. To sub-sample the same number of points from all curves, you can use the “lapply” function to apply the function and “n” parameter to each curve in the list.

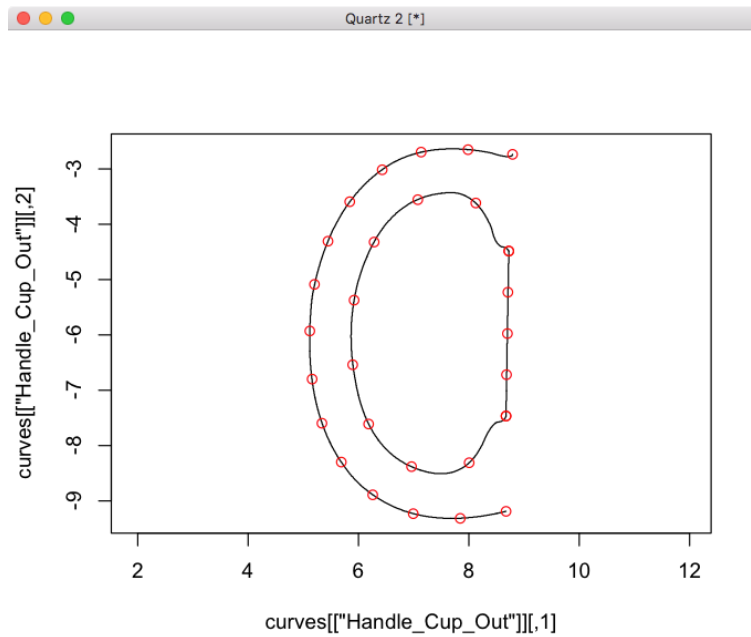
```
> curves_sub <- lapply(curves, pointsAtEvenSpacing, n=10)
```

5. To plot the curve points start by creating the plot window to a size that will accommodate all of the curves.

```
> plot(curves[['Handle_Cup_Out']], asp=1, type='n')
```

6. Plot the original curves with points at pixel resolution (to create a nice smooth curve) as a line and overlay the sub-sampled curve points as red points.

```
> lapply(curves, points, type='l')  
> lapply(curves_sub, points, col='red')
```



Plotting the curves of the coffee mug handle with sub-sampled curve points.

8 Shortcuts (keyboard and otherwise)

For large digitizing projects, there are a few shortcuts to make the digitizing faster and your life a bit easier.

1. The first are the keyboard shortcuts. The key “**x**” has the same actions as a double-click in the image frame. If the cursor is positioned over a marker (landmark, curve control point or ruler point), pressing “**x**” will select that marker and make it the current marker. The key “**n**” (next) will advance the current marker to the next in sequence. The order of advance will proceed in the same order as the points are listed in the control panel. The key “**p**” does just the opposite, changing the current marker to the previous marker in sequence.

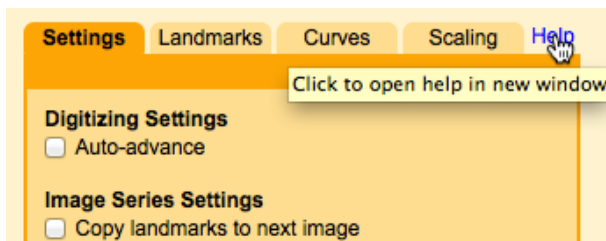
2. On the “Settings” tab you’ll see a checkbox marked **Auto-advance**. Checking this box will cause the current marker to change to the next in sequence after each marker is placed (through a double-click or by pressing “x”). This way you can position a whole set of landmarks or curve control points using a single click per point. You can always use “p” to back up and fix a previous marker. All the checkbox settings use cookies in the web browser so that any settings you have set during one digitizing session should be saved in subsequent sessions.

3. On the “Settings” tab you’ll also see **Image Series Settings**. These apply when you load a folder of images or multiple images into the digitizing app at the same time. Each of them will copy the indicated elements (landmarks, curves, etc.) to the next image you open (using the “Next Image” and “Previous Image” buttons). Note, the elements will only be copied if that element is not already saved in the next file. So if the next image already has saved landmarks associated with it, the landmarks will not be copied over to the next image. Also, any copied elements will not be automatically saved to the next image. A warning message will appear that you have unsaved changes if you try to change the image or exit without saving the copied elements. Lastly, you can modify any of the copied elements once they have been copied to the next image.

This can come in handy when the scaling for all images is the same. You can find the scaling in the first image and then copy the scaling to each image after that. This is also handy when digitizing landmarks on a series of consecutive images from a video. The landmarks will change little from frame to frame so they can be copied and then adjusted in each image to reflect the incremental changes.

9 Where to go for help

In the top right corner of the app you'll find a link "Help" that will open a help reference page in a new tab. This provides much of the same content in this tutorial but in a form more convenient to reference while using the app.



Open a help reference in a new tab.

Additionally, please direct any questions not addressed here and report any bugs found to aarolsen@gmail.com.

10 Citing StereoMorph

The StereoMorph R package and its associated digitizing app are the result of several years of work. I am pleased to offer the app open-source and free of charge and hope that users find it useful and that it brings about interesting, fruitful and fun advances for people both within and outside the field of morphometrics. I only ask that if you use the digitizing app and share your results that you cite StereoMorph. For peer-reviewed publications, please cite the [following article](#):

Olsen, A.M. and M.W. Westneat. 2015. StereoMorph: an R package for the collection of 3D landmarks and curves using a stereo camera set-up. *Methods in Ecology and Evolution*. 6:351-356. DOI: 10.1111/2041-210X.12326.