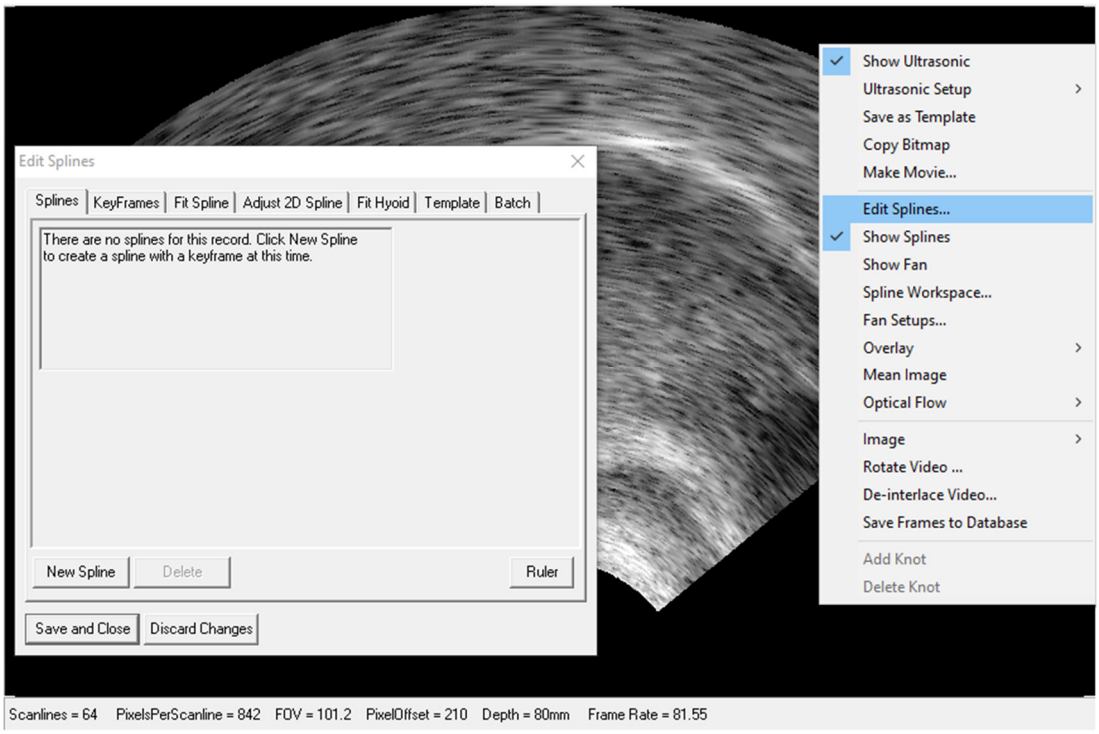
QuickStart: Automatically spline recordings

The Edit Splines dialog is the interface for creating and editing splines based on either ultrasonic or video data. To access the Edit Splines dialog,  right-click on the ultrasonic display in any task window or the video display.



Note

Ultrasonic and video data streams are stored and processed separately and when splines are generated they are associated with either video or ultrasonic. This prevents lip video splines appearing on an ultrasonic tongue image and vice versa.

DeepLabCut Batch Process

It is recommended to use the fully automatic DeepLabCut Batch processing to spline every frame for every recording in the session that has just been recorded. There are different models for splining tongue contours or front facing lip camera.

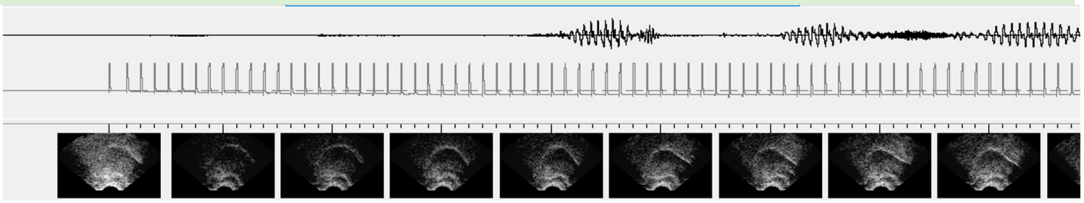
- If you have an ordinary CPU processor then this might take as long as 20 times real time (i.e. 30 minutes of recorded might take 10 hours to process).
- For a faster CPU processor more like 8 x real-time (i.e. 4 hours for 30 minutes of data)
- If you have an NVIDIA GPU configured then more like 2 x real-time (i.e. 1 hour for 30 minutes of data).

This process will estimate the contour at the timepoint when the ultrasonic/video frame is generated and the spline will be linearly interpolated for time points in between.

DeepLabCut Batch Tongue contour estimation

1. Use the batch sync process (see separate section) to make sure all your recordings are synchronised for audio-video and audio-ultrasonic.

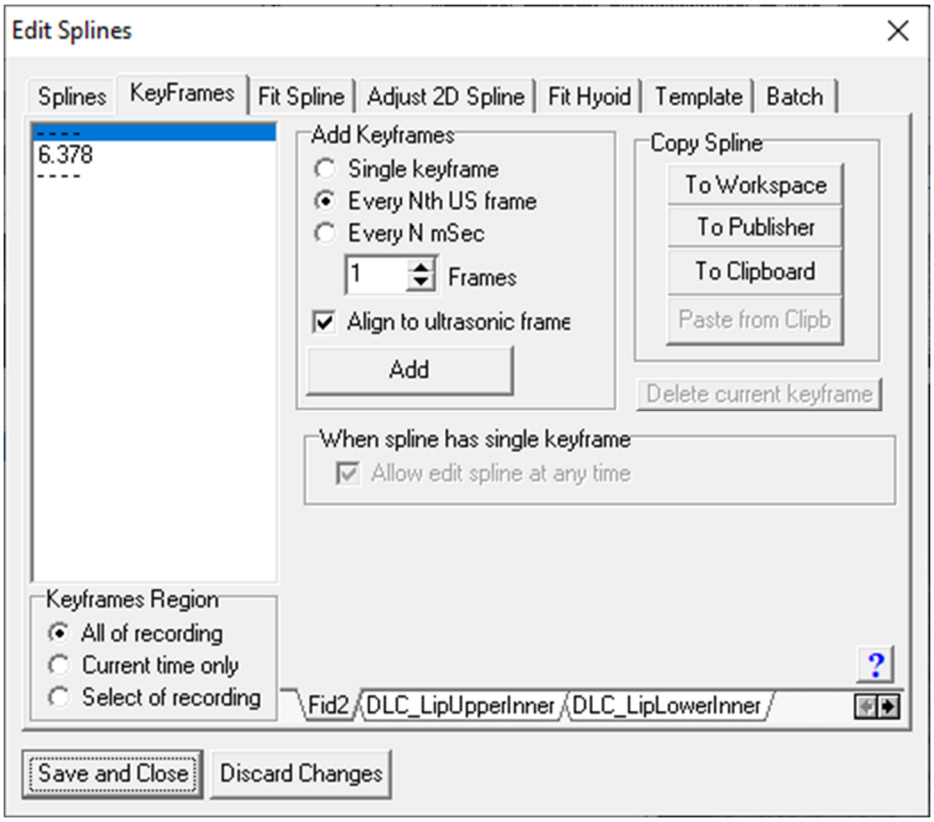
> Note
>
> Ultrasonic data is only synchronised when viewed for the first time and video data is not automatically synchronised. Both require explicit batch synchronisation to be sure.



Tick marks above ultrasonic chart should line up with pulses on channel 2.

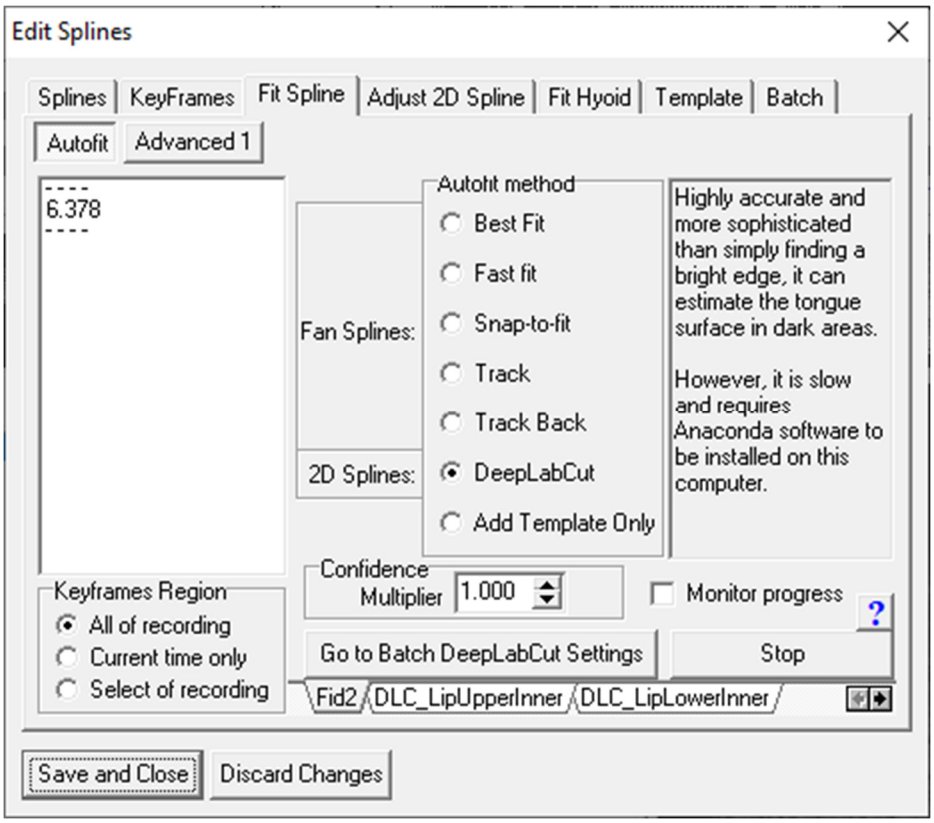
2. Select the **Keyframes** tab.

   By default it should have the **Add Keyframes** settings shown. Every Nth keyframe, Frames = 1, Align to ultrasonic.



3. Select the **Fit Spline** tab

By default the Autofit method should be set to **DeepLabCut** and the keyframes region should be set to **All of recording**.

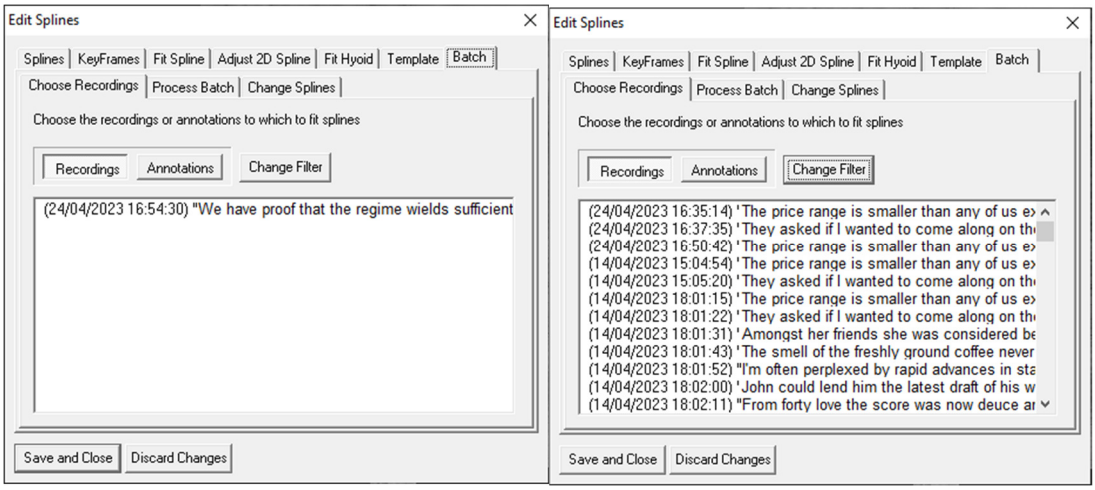


4. Select the **Batch tab**

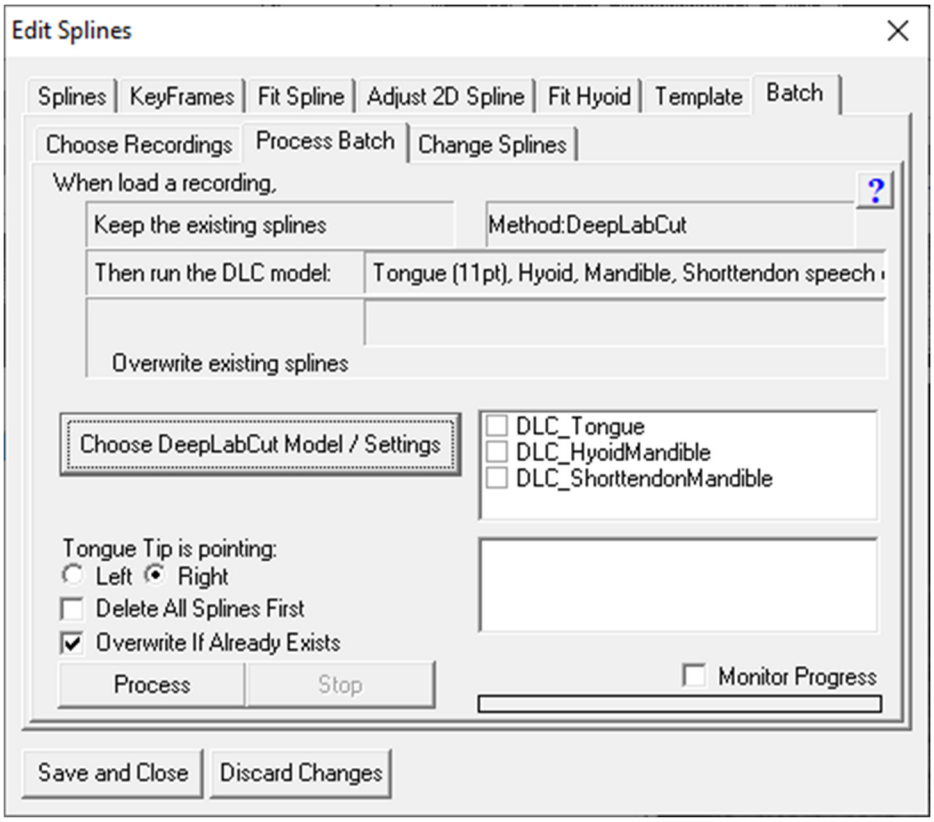
The Go to Batch DeepLabCut Settings button will take you to the Batch tab (or simply click the Batch tab.). DeepLabCut defaults to the best tongue model but if you want to change the model there is a button on the Batch tab to allow you to do this.

Under the Choose Recordings subtab make sure the Recordings button is selected and click the Change Filter button. Then set the filter to Current client and **All recordings.** The box will then show all the recordings for the current client.

a. Select the **Choose recordings** subtab
   Leave the selection on the Recordings Button so that whole recordings are processed. Set **Change Filter** to **All recordings**.

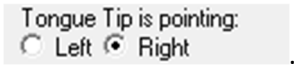


b. Select the Process Batch subtab.



Check that the specification at the top confirms

- you are using the DeepLabCut method
- You are using the Tongue model if ultrasonic or Lip model if lip video

If you have recorded the ultrasonic data with the tongue tip pointing left then change the radio button 

. Check all the spline options DLC_Tongue, DLC_HyoidMandible, DLC_ShorttendonMandible.

If you check 

The ultrasonic display will show the estimate for every frame as it is calculated. This will slow down the splining hugely but you can switch this off while the splining is in progress so it can be useful to check that the estimation is working as you expect.

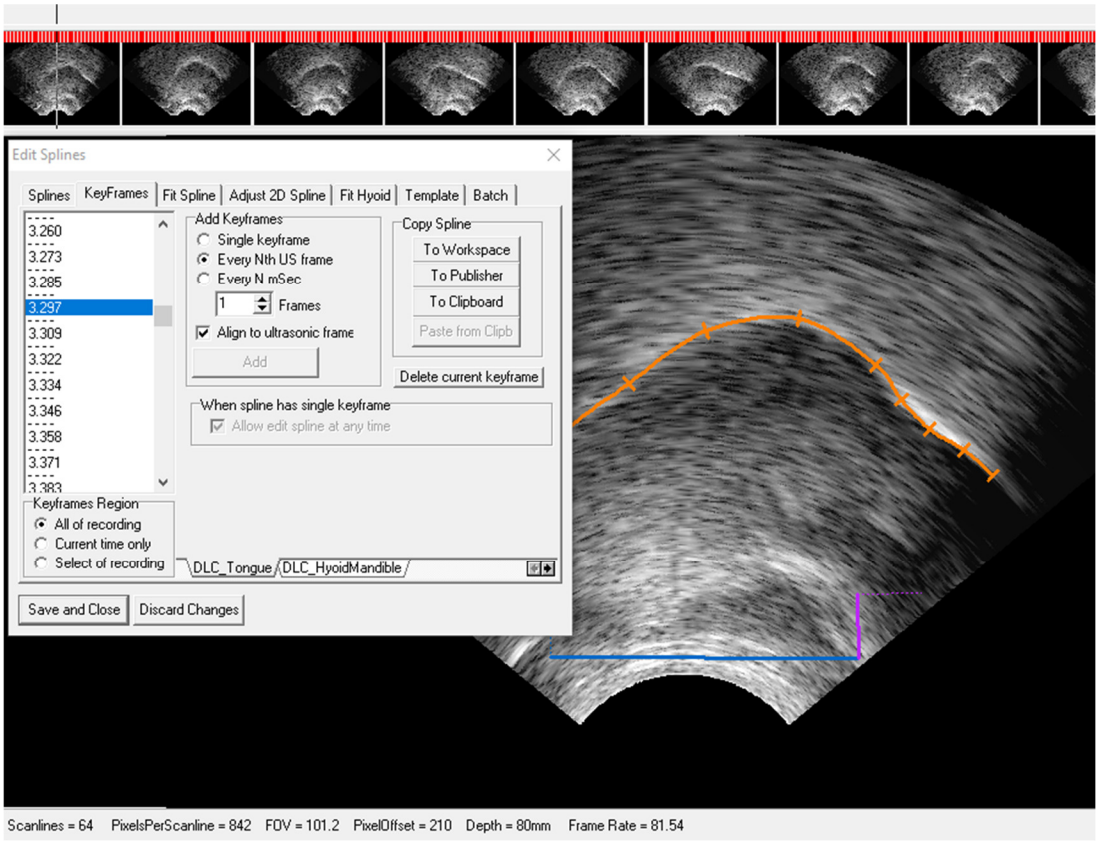
Click 

to start the automatic fitting.

Hand correcting 2D (DLC) splines

DLC splines are freeform splines with no restriction in shape. They are referred to as 2D splines in AAA. It is NOT RECOMMENDED TO MANUALLY CORRECT DLC SPLINES as the model can distinguish between bright edges related o the Tongue contour and others which are artifacts or other structures such as the Epiglottis. THIS SECTION ON HAND CORRECTING CAN BE IGNORED.

After automatic splining is complete, every ultrasound frame or video frame will have a spline **keyframe** with a unique spline. It is possible to manually edit these from the **keyframe** tab.



To select a single keyframe:

1. Make sure the 

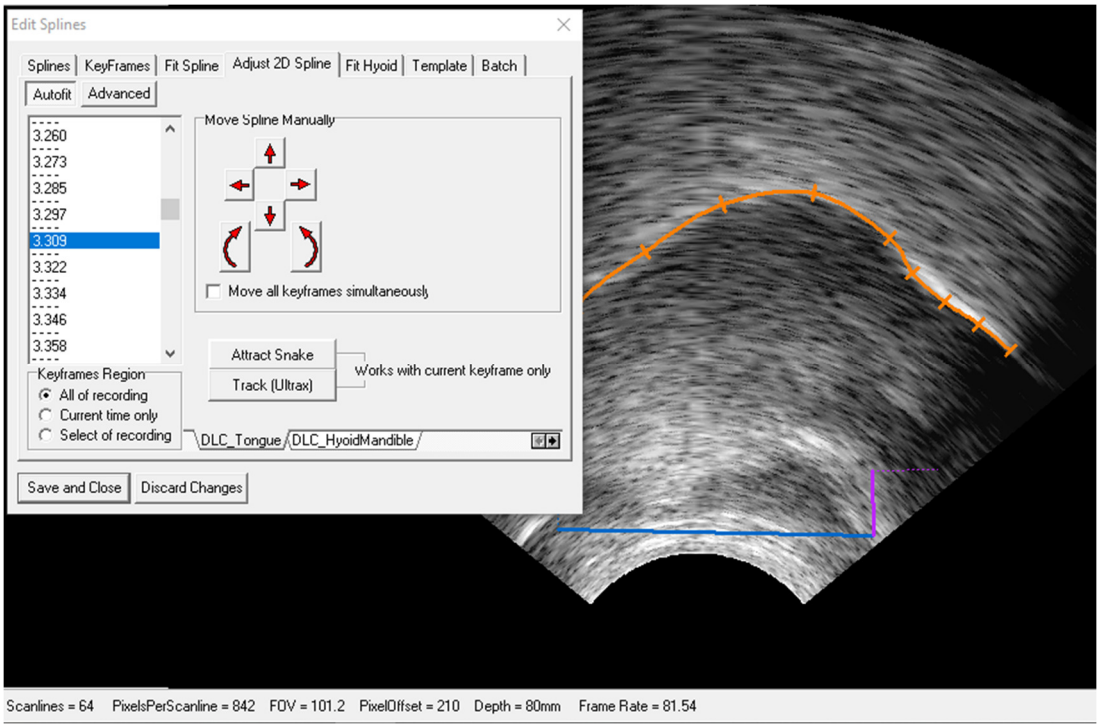
spline tab is selected. Use the  buttons if it is not visible.
2. Click on a time in the list on the left. The spline will then turn orange to indicate that it has been selected.
3. The DLC_Tongue spline is controlled by 11 **knots** (control points) represented by ticks along the contour. Click near a tick and drag to move it around.

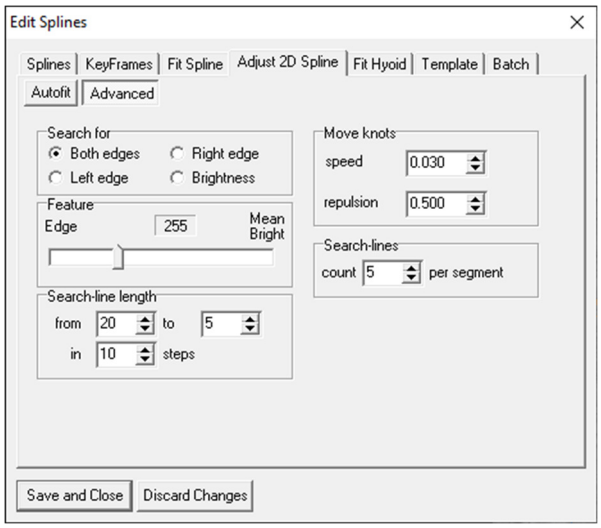
Tip

If you hold down the <ctrl> key, then click and drag on the ultrasonic display, a contour will be drawn with the same number and relative spacing of knots, starting where you first click and ending where the drag ends. You can use this to hand draw the entire contour.

Moving and rotating 2D splines

The **Adjust 2D Spline** tab offers controls to translate and rotate whole splines. This is not very useful for tongue or lip splines and more often applied to fiducials (see Fiducial Spline section). There is also a snakes algorithm. This is intended to iteratively adjust the selected 2D spline to fit to the nearest bright edge. THE SNAKES FUNCTION IS NOT RECOMMENDED. Used sparingly (clicked a couple of times) it may move the spline closer to a bright edge but extended use will distort the spline shape. See Appendix A for description of Algorithm. The Advanced button reveals the parameters for adjusting the Snake algorithm behaviour. But this bright edge hunting algorithm is inferior to DeepLabCut and discouraged. The Track Ultrax button has been deactivated.

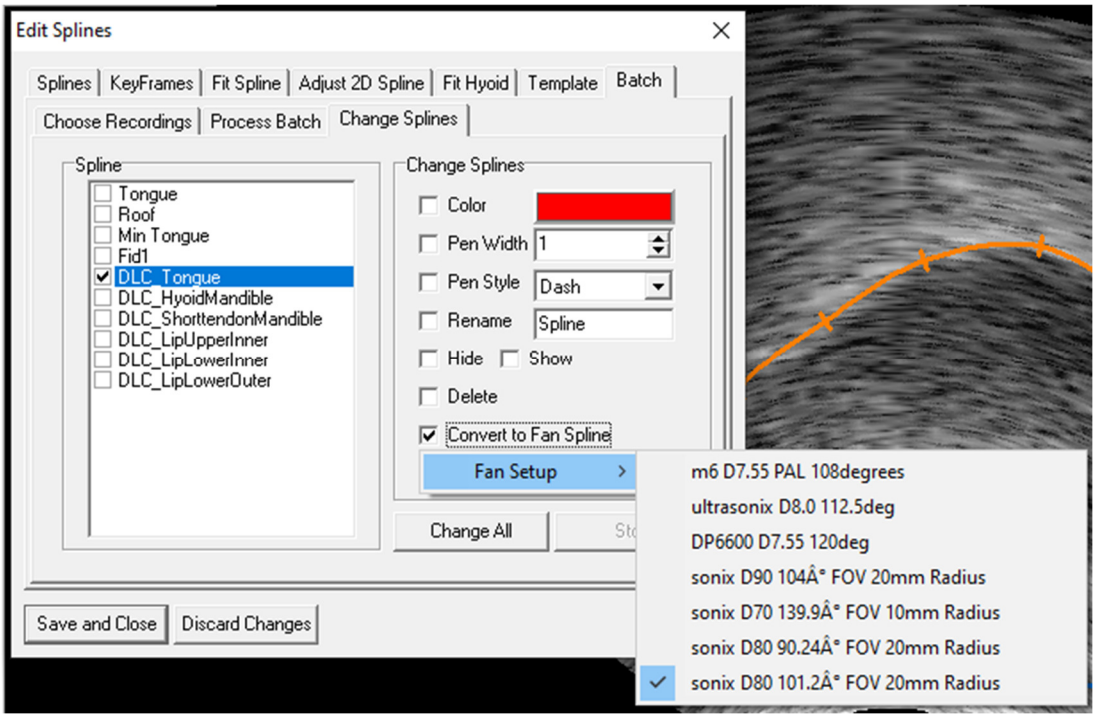




Converting 2D splines to fan splines

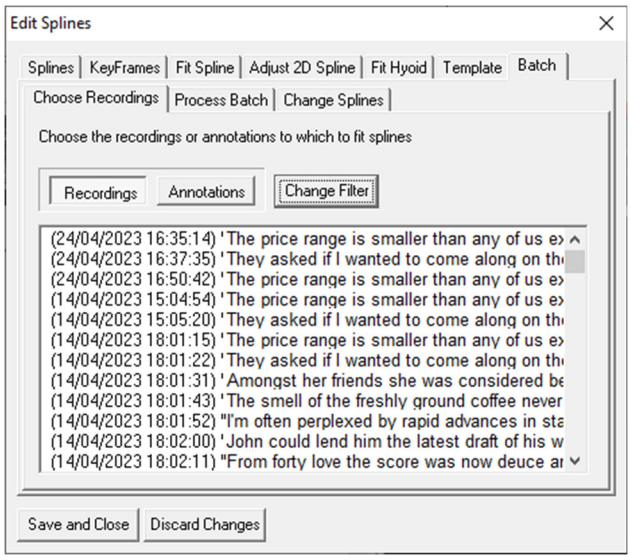
THIS IS NOT A RECOMMENDED PROCESS as there are better ways to analyse splines but it provides a route to exporting polar coordinates.

1. Click on the **Batch** tab then the **Change Splines** subtab
2. Select the DLC Tongue spline
3. Select Convert to Fan Spline
4. A submenu will appear with preformed fans with different origins and extents. The fan designed to fit the ultrasonic data with origin at the probe origin.

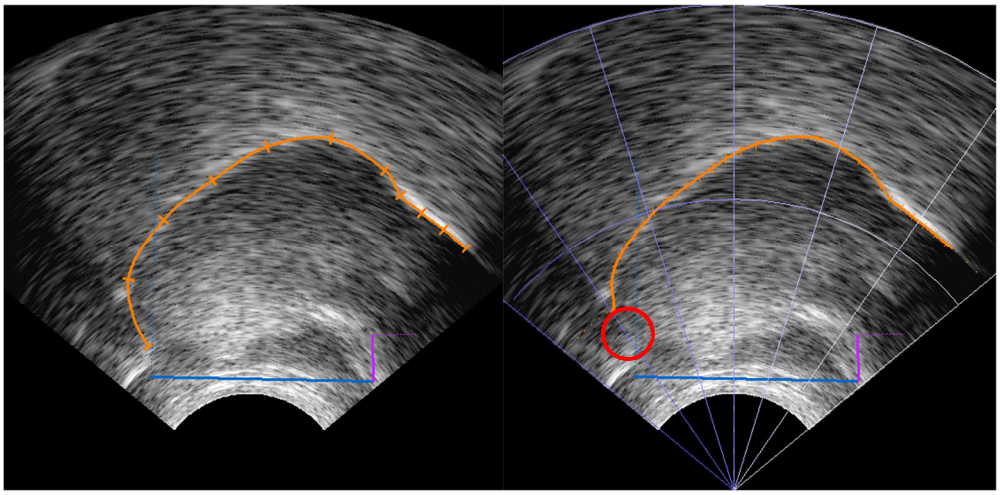


5. Check the Choose recordings tab has selected the recordings that you want to convert. Use the 

to change the selection.



6. Return to the **Change Splines** subtab and click 



Note that choosing this fan origin can result in loss of tongue contour where it is parallel to the fan axis (red circle above).

7. Instead, create a new fan with its origin at the short tendon. This will guarantee the whole contour to be converted.  Right-click and choose **fan setup.**
8. Change the fan name to Short tendon. Set the cyan length measure from top to bottom of display and set length to ultrasonic depth + probe radius. Make sure the pharynx and dental labels are the right way round. If not, use 

button. Click and drag the fan origin to the short tendon.

Appendix A

SNAKE spline fitting algorithm

2D splines

A *cubic spline* passes through several knots. There are no extra control points (as with *B-Splines*) - we

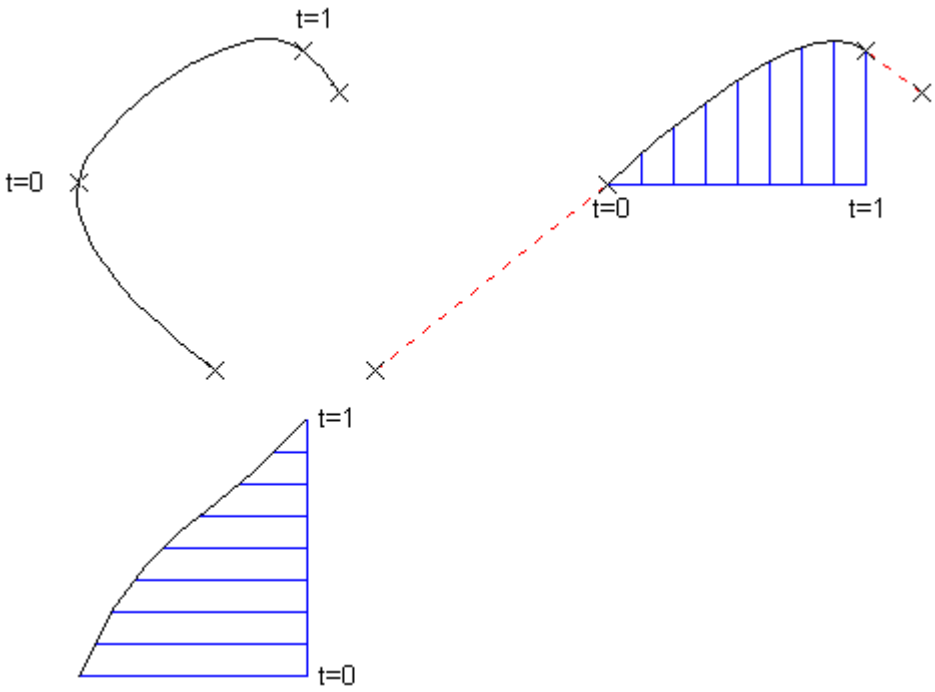
feel that controlling the spline simply by the positions of the knots is more intuitive for users (who set the initial position of the spline) and is easier for any Snake algorithm.

The line is composed of several segments - each segment runs between two adjacent knots. The segment is drawn with "time" t running from 0 to 1

$$x = ax*t3 + bx*t2 + cx*t + dx$$

$$y = ay*t3 + by*t2 + cy*t + dy$$

The coefficients (a,b,c,d) for the line between knot N and knot N+1 are calculated independently for the x-coordinates of the line and the y-coordinates.



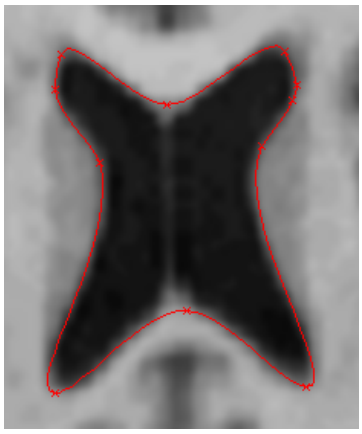
The coefficients are calculated so that:

1. the line passes through the two knots N and N+1
2. the slopes at knots N and N+1 are specified

The slope at knot M (dx/dt or dy/dt) is the slope of a quadratic which passes through knots M-1, M and M+1

If two knots have no neighbour to the left or right then a quadratic expression is used for the segment. If two knots have no neighbours to both the left and right then a linear expression is used for the segment.

This form of spline has been found to be well behaved and is intuitively easy to use. It is not completely rotationally invariant but the variation is negligible.



(11 knots)

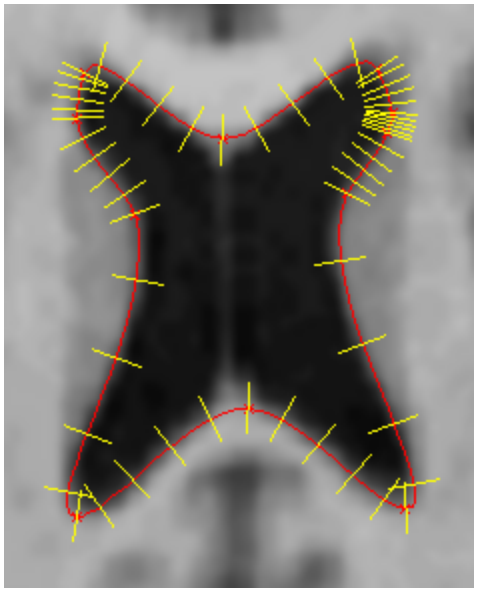
There are several advantages to using this form of spline to represent a Snake. The coefficients and the line are quick to calculate. There are only knots, not knots and control points. If the snake "knows" that a line segment should increase its curvature and move up, then which way should it move the knots and the controls points or a B-spline or Bezier? We have only knots and the line reacts sensibly to changes in the position of a knot.

Snake

A Snakes algorithm moves the knots so that the line gets closer to a "feature". We are interested in edges so a "feature" is an edge. It could equally well be a particular brightness, a dot, the end of a line or a combination of features.

The algorithm is iterative. In each iteration, the algorithm must decide where to move each knot: how far and in which direction.

To move knot N, search-lines are drawn normal to the tangent to the spline. A search-line is drawn through the knot and w search-lines are drawn on either side equally spaced between knot N and knots N-1 and N+1. In this example, w is 5 and the search-lines extend h = 20 pixels on either side of the spline.



The pixels lying under each search-line are examined working from one end of the search-line to the other:

p is the brightness of the pixel being examined

q is the brightness of the previous pixel

We are looking for edges so we calculate a number which is bigger if there is an edge near the the spline:

left-rising	c := + (p-q)*sign(ay)
right-rising	c := - (p-q)*sign(ay)
either	c := abs(p-q)*sign(ay)

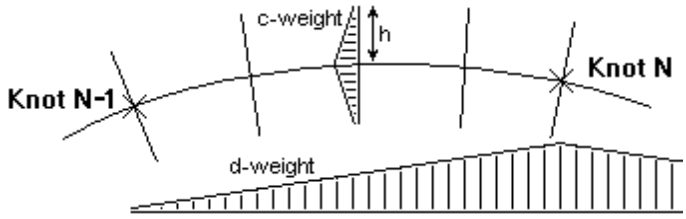
ay is the distance along the search-line from the central spline

If p is different from q then there is an edge. The edge can be "left-rising", "right-rising" or either. The snake will hug "internal" boundaries, "external" boundaries or both.

We calculate a weighted value of c

weight = (h-abs(ay))/h;

the weight is 1 on the spline and decreases linearly to zero at the ends of the search-line.



The image does not have to be smoothed. The above calculation automatically smooths the contributions from different pixel-pairs.

Each (weighted) value of c contributes to the movement of Knot N. If the sum of c is positive, it implies shifting the line to the "left". So the origin of the search-line should move "left" along a vector d. The length of d is given by c, the direction of d is the direction of the search-line.

The d values are vector-summed to give an overall movement for Knot N. This is a weighted sum: the weight falls off linearly with distance from Knot N and is zero at Knots N-1 and N+1.

The weighted sum of d is scaled by a "speed" factor which controls how fast the Knot moves in each iteration. Typical values range from 0.005 to 0.05.

The algorithm is iterated several times (typically 10 to 100). At first, the search-lines are long (h = 20) which gives a coarse movement of the knots but has a large capture distance. Later iterations have smaller search-lines (h = 5).

Knots move along the spline

With successive iterations, the knots move around. They move the spline from side to side and they move along the spline. You may require several iterations of the above (iterated) algorithm to see the effect.

If two knots get too close to each other (within one or two pixels) then the spline goes crazy. A term can be added to force pixels apart. This force should only act over small distances and should be negligible above (say) 10 pixels. A suitable force involves moving each knot away from each neighbour (N+1, N-1) by a distance k/d (d is the distance to the neighbour; k is a speed constant, e.g. 0.3).

You could restrict each knot only to move along the search-line that passes through the knot: so the spline moves from side to side but the knots don't move along the spline. The knots then tend to stay spaced-out as they were originally placed. The knots generally, move along the spline to sensible positions, so it usually isn't neccessary to force them to retain their original spacing. However, if the spline is not closed then the end-points can occasionally wander off. We have restricted the movement of the endpoints of the spline so that they can only move along their searchlines and not move so as to extend or shrink the overall distance between the endponts.

Some snake algorithms introduce other internal forces into the snake:

- tension: pulls the knots together like a rubber band
- stiffness: straightens the spline

For our application, both tension and stiffness do more harm than good.

Rather than search for edges, we can search for pixels of a particular brightness:

brightness c := -abs(r-p) *ay/h /10

r is the "required" brightness

(the "/10" constant brings the value of c into the same range as the edge calculation)

A combination of 70% edge detection and 30% brightness detection seems to work well. (The required brightness is the mean of the pixels under the spline.) The snake follows a strong edge but where the edge becomes ill-defined, it follows the mean brightness.