

Aegisub-Motion User Handbook

November 5, 2011

Contents

1	Preamble	2
2	Trimming the Video	2
2.1	AviSynth to x264	2
2.2	Just x264	3
2.3	Extra Info	3
3	Using Mocha	4
3.1	Starting a New Project	4
3.2	Motion Tracking	4
3.3	Exporting the Motion Data	5
4	Using Aegisub-Motion.lua	6
4.1	Choosing the Lines	6
4.2	Gathering Information	6
4.3	Caveats	7
4.4	Aegisub-Motion's Interface	8

1 Preamble

This document is intended to explain the usage of Aegisub-Motion.lua, an Auto4 plugin for Aegisub that takes motion tracking data output from any motion tracking program capable of outputting the correct format (“Adobe After Effects 6.0 Keyframe Data”) and converts it into ASS v4+ override tags.

This guide covers the entire process I use when motion tracking a scene. If you already are accustomed to using motion tracking software, it’s probably relatively safe to skip the next two sections entirely and just read about how to use the plugin itself; however, because Aegisub-Motion is written with some inherent assumptions that people using it will roughly follow the workflow described below, you should probably at least skim the text to verify that the way you work won’t cause the script to break (though to be honest, anyone who regularly does motion tracking for subtitles and doesn’t already have a script to convert their output for them is probably insane). I’ve tried to make the script fairly robust, but I’m sure it’s still quite easy to do something or other and make it not work.

2 Trimming the Video

The first thing that you have to do (assuming your video is not currently quicktime-decodable) is to trim out the scene of the video that has the sign/object that you want to motion track in it. Mocha uses quicktime to decode H.264, so make sure you have that installed. I know of two simple methods to trim out the scene. The first involves feeding an AviSynth script into x264, and the second just requires x264. I usually use avisynth just out of habit, but if you don’t already have it on your computer (or you don’t have windows/wine), you can save yourself the hassle of installing it.

2.1 AviSynth to x264

This section assumes that you are familiar with basic AviSynth usage.

The first thing to do is figure out the starting frame of the section you need to track. I usually do this with aegisub because I usually time the lines to the scenes they go in first. After that it’s a simple matter of pulling the start and end frame for the line (which for now we’ll assume spans the whole scene) from Aegisub’s video display. In Fig. 1, you can see that Aegisub displays the frame number of the current video frame. It is possible to simply copy this value out. Once you have collected both the start and end frames, you can create a simple avisynth script.

```
FFVideoSource("C:\ep2-720-8bit.mkv").trim(15771,15880)
```

Now it is a simple matter of feeding this into x264. Using avisynth significantly simplifies the necessary x264 command line.

```
C:\x264>x264.exe --preset veryfast --profile high --crf 22 -o  
C:\onbreak.mp4 C:\onbreak.avs
```

Using AviSynth eliminates the need to specify the seek and the number of frames to encode. Also, because AviSynth doesn’t understand vfr, it eliminates the need to specify the framerate.



Figure 1: The current video frame in Aegisub's video window.

2.2 Just x264

Quicktime seems to append (an) extra frame(s) on the end of mp4s trimmed with x264 when decoding them. This can be dealt with in mocha.

As with the AviSynth method, the first thing to do is determine the starting and ending frames from the video window as shown in Fig. 1. Once you have collected this data, you can construct your x264 command line.

In our example case, the start frame is 15771 and the end frame is 15880. With this information, we can construct our x264 command line.

```
C:\x264>x264.exe --preset veryfast --profile high --crf 22 --seek 15771
--frames 110 --fps 23.976 -o C:\onbreak.mp4 C:\ep2-720-8bit.mkv
```

Note that the value for `--frames` is `endframe - startframe + 1` because the frame numbers are inclusive. Also note that the output has to be an mp4 because Quicktime cannot split matroska files. Also note that the file *must be cfr* for mocha to be able to open it. The best way to be certain of this is to specify `--fps` in the command line. The framerate is irrelevant. You should be able to set it to anything you want. All that really matters is the number of frames.

2.3 Extra Info

Aegisub uses FFMpegSource2 as the video provider by default, and if your original source video is H.264 in Matroska, then x264 will use FFMS2 to decode the video. Accordingly, the example AviSynth script also uses FFMS2. This should cause all of these programs to decode the video in the same way, assuming they are all using the same FFMS2 version.

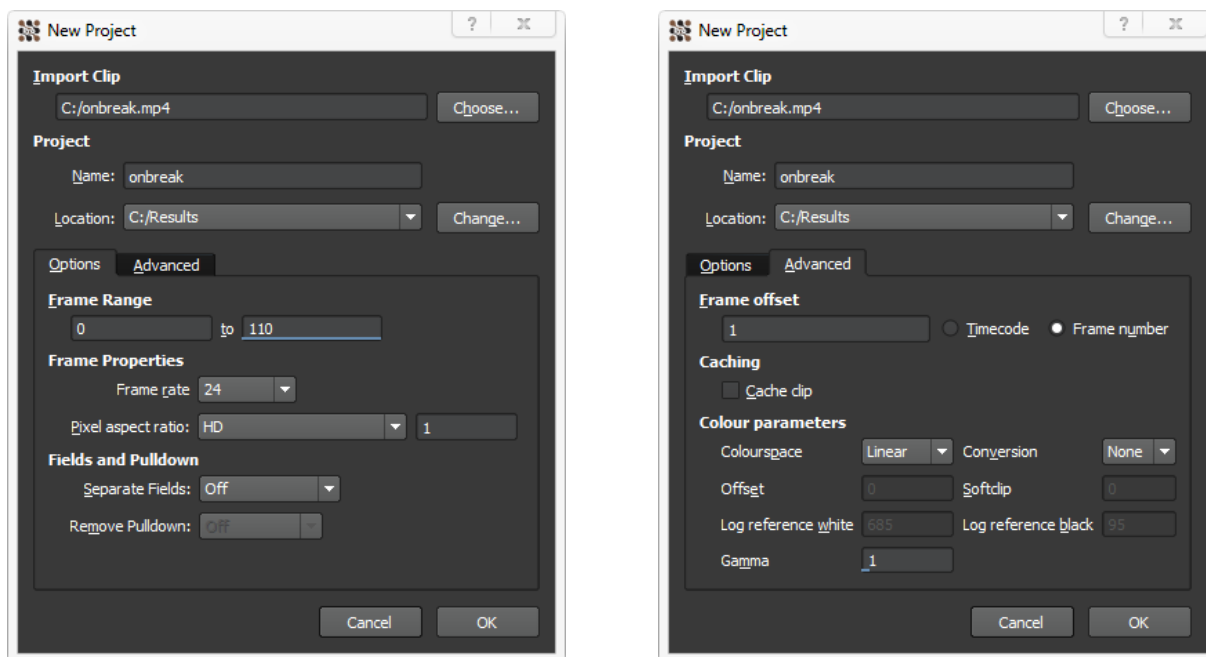


Figure 2: New Project—regular and advanced options.


3 Using Mocha

I’m only going to cover basic Mocha usage here, for two reasons. One, because I haven’t used many of its advanced features enough to have a firm grasp on what exactly they do, and two, because it would simply take too long to do otherwise. Imagineer Systems’s website has some great video tutorials on how to use the software. Those deal with everything from beginner basics to more advanced techniques, and I’d strongly recommend watching them.

3.1 Starting a New Project

Obviously, the first thing to do is to open a new project. If you can’t figure out how to do that on your own, you should probably just give up now. It’ll be less painful for both of us this way. Fig. 2 shows the new project dialogue. There isn’t that much to do here. Make sure the frame range matches the number of frames you cut. If you don’t know what you’re doing, the pixel aspect ratio should always be left at (or changed to) 1. If you’d prefer for Mocha to display the first frame as frame 0 instead of frame 1, change the frame offset under the advanced tab. This is an entirely aesthetic change, however, as it does not affect the frame numbering on the tracked data (as far as I know).

3.2 Motion Tracking

The next step is simply to create a layer spline. To do this, click the “Create X-Spline Layer Tool” . Click to create points around the object that you want to track. For most of the stuff I’ve done, rectangles work fine. After you have the shape you want, right click to exit the spline tool. You can adjust the points afterwards if you like. Once you’re done making the shape, make sure the “Link to track” option under “Layer Properties”

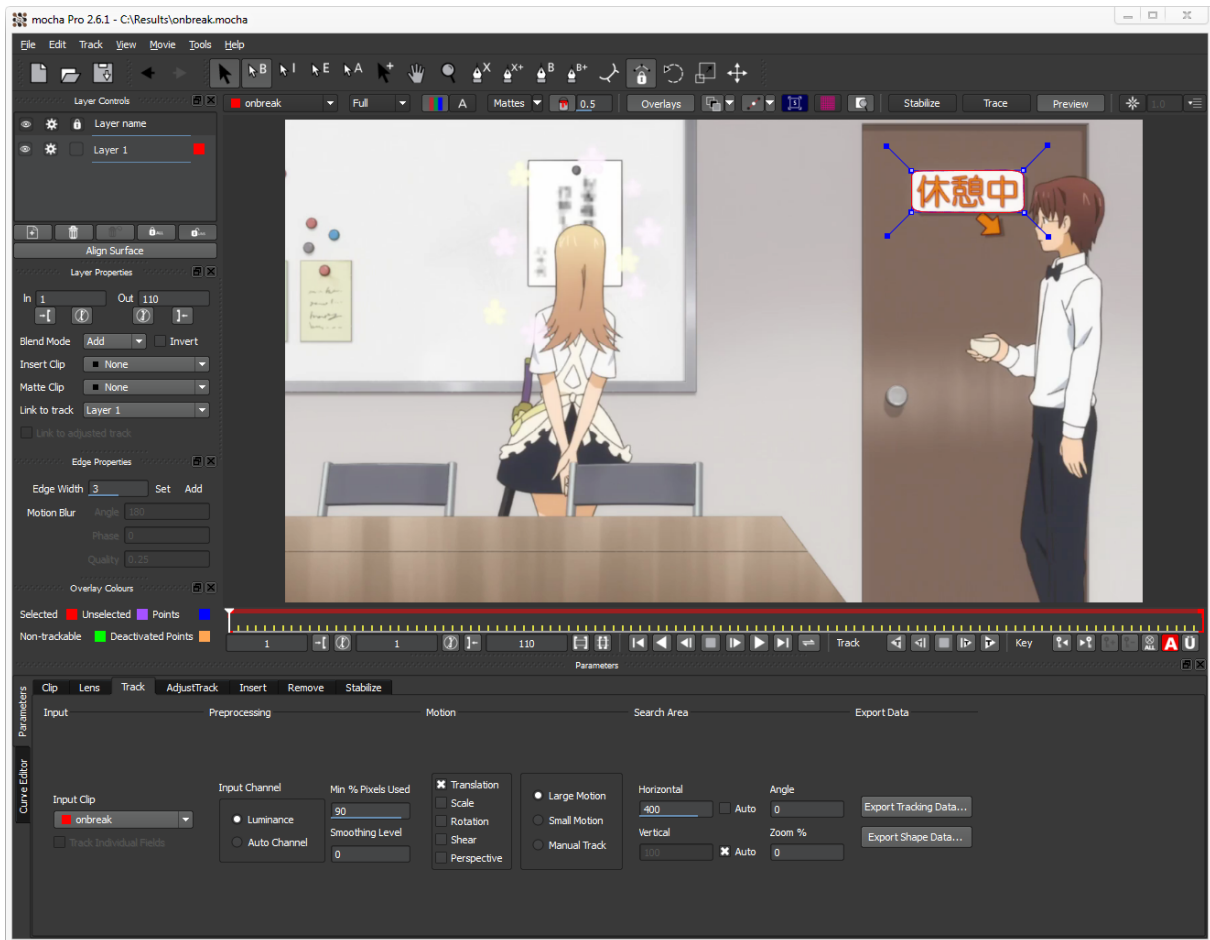



Figure 3: Mocha's main interface.

on the left side of the window has a layer specified (in this case, track one). No tracking data will be collected otherwise.

Click the track forward button , and the program will proceed to track the video under the spline. Depending on how fast your computer is and how long the clip is, this can take a while. It's a good idea to keep an eye on the program to make sure that it doesn't obviously mess up in the middle, which it does occasionally. Once it's done, play it back once or twice to see if the spline moves smoothly with the part you were tracking.

For people who know what they're doing: I'm sorry, but you cannot have keyframes in the middle of the tracked data because this will mess up the script.

3.3 Exporting the Motion Data

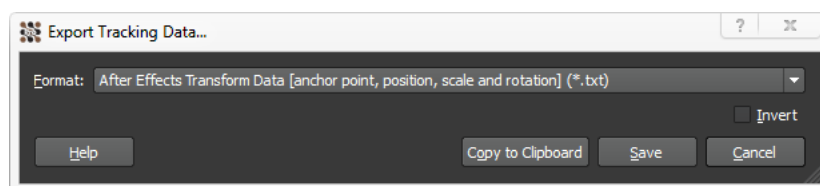


Figure 4: Exporting the tracking data.

Click the "Export Tracking Data..." button to open the save dialogue. A window

will pop up with a drop down list of formatting options. As shown in Fig. 4, select “After Effects Transform Data” as the type to save the file as (this is assuming you will be using my Aegisub-Motion.lua script to parse the data). You can either choose to save it to a file or to copy it to a clipboard. I often copy to clipboard and then paste it into notepad++ to save (though since it’s just a simple text file, either method should end up with the same results).

And that’s it. Assuming that you watched the tracking to make sure it assimilates well with the motion of the object you are trying to track, there is nothing more to be done in Mocha. Keep in mind that *the aegisub output is only as good as the tracking data*. Mocha has a lot of very advanced features that aren’t even touched on here, as well as options that can be tweaked to improved the accuracy of your results. I strongly recommend taking the time to actually gain a somewhat in-depth knowledge of the program if you plan on using it with any regularity.

4 Using Aegisub-Motion.lua

Aegisub-Motion.lua is a lua script that I’ve written for acting as a bridge between Mocha’s output and the ASSv4+ subtitle format. It is currently very much a work in progress, but it currently has support for applying position, scale and rotation data. The vast majority of signs I’ve run across that need to be motion tracked have only had position and scale changes, so I feel like these three options already cover all common-use cases. I have yet to look into the remaining two options that Mocha offers to track, shear and perspective, for two reasons. First, they are more complicated to implement, and I haven’t really looked over the available output formats for them in Mocha anyway. Second, because they have been nonexistent in my typesetting experience, I’d rather spend time adding more useful features and improving the parts of the script that will see common use. In the following subsections, I’m going to more or less discuss how the program works step-by-step in order to hopefully give the most complete explanation of what it can and cannot do.

4.1 Choosing the Lines

The script is written so that multiple lines that are all supposed to follow the same motion can all be set by running it once. Each individual line does not have to be the total length (in frames) of the motion tracking data, but the endframe of the latest line minus the startframe of the earliest line *must* be equivalent to the number of frames tracked. I can’t think of very many scenarios where the number of tracked frames would exceed the number of frames to which the tracked data need be applied. But just in case such a scenario is necessary, the capability exists.

4.2 Gathering Information

One thing you ought to know is that in order to run this script, you *must* have a video loaded in aegisub. It doesn’t matter if it’s a dummy video (though given the nature of this script it would really make sense if you had the actual video you were trying to typeset loaded), but the script runs a validation macro to see if there is a video loaded. The reason for this is simply because the script frequently uses functions to get frame times from the loaded video’s timecodes, and if there is no video loaded, it will simply error

out. Thus, it makes sense to disable the script entirely when there is no video, rather than allowing it to be run and simply error out immediately.

The script immediately gathers information about the style(s) of the selected lines, as well as the pertinent override tags in the line. It searches for (via simple regular expression) `\pos`, `\fscx`, `\fscy`, `\an` (alignment), `\bord` (and if that is not found, `\xbord` and `\ybord`), `\shad` (and as with `\bord`, subsequently for `\xshad` and `\yshad`), `\frz` and `\org`.

4.3 Caveats

To get proper results, it is very important to understand that these searches only find *the first given instance of each tag in each line* (with the exception of `\an` because VSfilter and libASS always use the last instance of it). This affects the tags that can be used multiple times throughout a line: border, shadow, scale, z-rotation. Take, for example, the following line: `{\fscx200\fscy200}This part is bigger. {\fscx50\fscy50}This part is smaller.` If you were to attempt to apply scale tracking data, the script would find the first `\fscx/y` and use their values (200 and 200) as if they applied to the whole line (rather than just the first half, as is the case). Furthermore, when actually running the script (and forgive me for getting a little ahead of myself here), the function that removes the tags from the original line will remove *all* instances of them from the line. You can see that this will obviously cause the tracked line to not appear at all as initially intended.

The solution to this is if a line needs to have multiple tags that will be changed by the application of the tracking data, that line should be split across multiple lines such that each line only has one of each tag. Furthermore, the override tags should all occur before the first character in the displayed line, as they are written at the very front of the line. This will ensure the proper output result, and from my experience, it is very rarely necessary (so I don't feel like it's too much effort to require).

Another important thing to note is that if you are going to apply position data, all of your selected lines *must* have a `\pos` tag. Any lines that do not will simply be skipped. The script will warn you about this after it has collected the override tag info from the selected lines. It is probably in your best interest to heed the warnings it displays.

There are two other things that the script will warn you about. The first is if your line lacks `\an5` alignment. The reason for this is that applying scale data to non-centered lines will simply not display correctly. If you ignore this, the script should still run normally, but I really can't recommend doing that. What I can recommend doing is setting a default style for all of the signs you are planning on motion tracking, and having that style's default alignment be centered.

The second warning occurs if the video resolution in your script header does not match that of the video currently loaded into Aegisub. This is making a bit of an assumption (namely that the resolution of the video you tracked matches the resolution of the loaded video), and it is probably the least important of the three warnings, as it can be easily fixed after applying the tracking information (by simply changing the header subtitle resolution). That said, the script resolution *does* need to be the same size as the resolution of the video clip you tracked the motion on, otherwise the result will not appear to be correct.

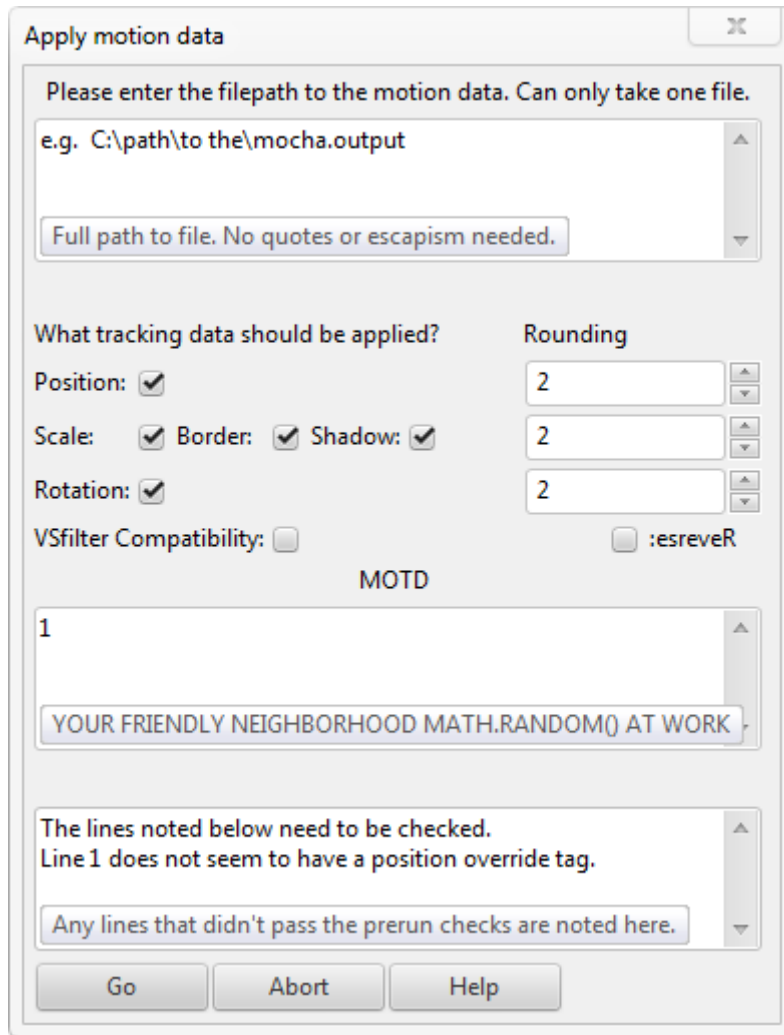


Figure 5: Aegisub-Motion’s interface, with tooltips shown.

4.4 Aegisub-Motion’s Interface

I have tried to make the interface as straightforward to use as possible, given the limitations of Aegisub’s automation interface capabilities. I feel that it is easy enough to understand that it does not merit a whole lot of explanation, but I will explain the features that may not be immediately understandable.

Fig. 5 is a picture of the interface. The first thing to note are the options. Here, you can choose what parts of your motion tracking data to apply. If you uncheck scale, neither border nor shadow changes will be applied, even if you leave their boxes checked. The integer editors under the “Rounding” header allow you to specify how many decimal places you want each attribute to have. I’ve found that for libASS usage, 2 decimal places is plenty. The allowed range for the rounding is 0–5 decimal places, but after two decimal places, the subpixel rendering is insignificant enough that it is pretty much impossible to tell the difference when the line is being played back. I’d say that anything greater than 2 decimal places is placebo.

The box below that contains the MOTD. Truthfully, it’s the box that was initially intended to take the path to the shear/perspective data, but since I haven’t implemented that yet and it is unlikely that I will do so anytime soon, it has been appropriated for the purpose of spitting out useless messages (although it seems like lua’s `math.random()`

function is severely broken, thus making this doubly useless).

Below that box are two more checkboxes. I will discuss them more in depth because they The first enables vsfilter compatibility mode. This is important, because, as I have stated previously, VSfilter only accepts integer values for `\fscx` and `\fscy`. This has the tendency to make scaled lines move very jerkily when they are tracked normally. Instead of inserting float scales into the lines, VSfilter compatibility mode operates on staged transforms. This allows vsfilter to render non-integer scales. VSfilter compatibility mode also locks down the round values to 2 decimal places for position and 2 decimal places for scale. The reason for this is that VSfilter only has 8 subpixel divisions per pixel, making higher accuracy entirely pointless. Rotation and scale are locked at 2 decimal places because having variable rounding with the staged transform algorithm would be annoying, and if I'm going to lock down two of the roundings, I might as well restrict the third one as well. Though, really, as mentioned above, anything above 2 decimal places is placebo anyway, so it's not all that bad.

The other checkbox is labeled "Reverse" (in reverse, get it hahahaha. Yes, I do hate myself). It makes the script reverse the order of the motion data and iterate backwards over each line (from the end to the beginning). This is useful in situations where a line starts offscreen and moves onscreen. Because it is hard to correctly set offscreen signs, this options allows you to set a sign as it should appear on its final frame and track its motion backwards, without having to change anything.

The final interface item is the warning box. As you can see in Fig. 5, it displays any necessary warnings about which lines may be formatted incorrectly. These were all covered in depth in the caveats section above, so I'm not going to talk about them any more here.

Hopefully that's covered everything. This will be updated with any major changes to the script.

Acknowledgements

I'd like to thank `\fake`, `--ar`, Hale, delamoo, nullx and zanget for their assistance, technical or otherwise, in writing this script, as well as jfs and Plorkyeran (among others) for the work they've done on Aegisub.