Automatic Tracking of Actin Filament Movement: *The Motility Tracking Programs*

Donald A. Winkelmann
Department of Pathology
and Laboratory Medicine
Robert Wood Johnson Medical School
University of Medicine and Dentistry of New Jersey
Piscataway, NJ 08854

Email: daw@pleiad.umdnj.edu
WEB page: http://simba.umdnj.edu/~daw

Introduction

Since their inception, *in vitro* motility assays have been crucial to the characterization of the properties of molecular motors. In the case of the myosin family of motors, global information on the dynamics of motors can be obtained from the observation of actin filaments moving on myosin coated surfaces using a fluorescent light microscope. Numerous variations on this sliding filament *in vitro* motility assay have been reported (Ishijima and others 1994; Kron and Spudich 1986; Sellers and others 1993; Uyeda and others 1990; Winkelmann and others 1995). This approach is very popular because it is simple and avoids many of the technical problems and the expense related to the manipulation and measurement of a few molecules in e.g. laser trapping experiments. However, the sliding filament motility assay introduces other unique problems of it own such as: the control of the myosin surface, homogeneity of the distribution of the myosin molecules, defects in the motor surface resulting from non-functional myosin heads, and dependence of the measured activity on the preparation protocol. Furthermore, the information content of video images is enormous, and it is often difficult to evaluate all but a small sampling of the video content. It is this last problem that prompted the development of the motility tracking programs described here.

Typically, anywhere from 10-100 or more actin filaments ranging in size from less than 1 μm to greater than 10 μm long are moving in every direction in a 80 μm by 60 μm microscopic field and recorded in a video stream at 30 frames/sec. The experimental video is often noisy adding to the difficulty in analyzing the motion of the filaments. One would like to automatically track a large number of these filaments, define the trajectory of individual filaments, determine the instananeous velocity of the filaments moving along a trajectory, and record the filament length, the distance of continuous motion and the duration of pauses in the movement. To accomplish all of this we have developed a set of programs collectively called the 'motility tracking programs' to retrieve this information from noisy video microscopy data (Bourdieu and others 1995; Kinose and others 1996).

Briefly, video segments are digitized and recorded directly to disk with a video capture card and image analysis software. Alternatively, video can be recorded with a frame accurate SVHS VCR and digitied after the experiment. The analysis of the digital video proceeds in a number of steps. First, a single representative image is displayed so that adjustable parameters for filtering the image can be set (*setfilters*). This is crucial because of the video data is noisy and there are changes in illumination, contrast, and background noise from scene to scene and between experiments. The acquisition program (*iplworms*) then uses these parameters and goes through every frame, extracting the objects defined by the filters. The acquisition program thresholds and filters the image, then creates a binary representation of the scene, and finally creates a database of the position, and shape of each filament. A frame with anywhere from 10-100 filaments is compressed to ~4 Kbytes and 100 frames are packed in a single file (a *cwb* file). We typically store ~1000 frames (~10 *cwb* files) per dataset.

A set of programs then analyzes the packed filament dataset (*cwb files*). Two programs (*threadworms and untangle)* read the files and analyze how the filaments in every frame superimpose with those in the previous frame, so that the continuity in time can be established; a label is then assigned to each filament, and this label is propagated in time. Unfortunately, these programs cannot resolve ambiguities when two filaments collide, so both are assigned new labels after the collision. The lifetime of a given label is established and only the longer ones (defined by an adjustable parameter) are kept and saved in individual files (by the program, *makewormlist*). Thus, each file contains the shape of a single filament propagated over a range of frames.

A key program then reads and interprets the single-filament files (*asciibatchns*). First, the shapes of the filament for all time points are superimposed, and a skeleton of the resulting superposition is defined and parameterized in arc length forming the 'track'. Many very short tracks are discarded at this point. For each

frame, every pixel in the filament is projected onto its track; the minimum, maximum, and mean arc length coordinates are then computed for that filament for each time point. This is done so that, as a filament turns around a sharp corner, an unbiased estimate of the reptation speed of the filament is computed. The output of this program is a file containing mean arc length positions along the track versus time, and includes the filament length. These data are then viewed and edited with an interactive graphics display program (*Speed Editor*) and the edited speeds saved in a *data* file. The *data* file contains hundreds of individual measurements that can be readily accessed with commercial spreadsheet software, analyzed, plotted and interpreted.

This document describes how the Motility Tracking Programs are used to analyze video datasets and provides details including the source code of the programs. This software is intended to aid with the interpretation of motility assay data and is provided '*as is*' with no guarantee of its suitability for any purpose except experimental analysis of data. The authors assume no responsibility or liability for loss or damages resulting from the use of this software.

Acknowledgements

The development of these programs was done as a collaborative effort of a number of investigators. Key to the effort was the interest and support of Professor Albert Libchaber, formerly of Princeton University and the NEC Research Institute and currently at Rockefeller University. People associated with his research group contributed enormously to the effort, especially Marcello O. Magnasco (M.O.M) of the Rockefeller University who was responsible for creating the algorithms and writing and testing the first versions of the code for most of the programs. Many others contributed to the development and testing of the ideas and software including: Laurent Bourdieu, Albrecht Ott, Tom Duke and Stan Leibler. Porting of the software to OpenGL, and development of the current versions of the programs and documentation was initiated by Qun Wang and completed by D. Winkelmann. Support for the development of this software was provided by the NSF (A. Libchaber), NIH (AR38454; DAW) and New Jersey Commission on Science and Technology (DAW).

References

Bourdieu L, Magnasco MO, Winkelmann DA, Libchaber A. 1995. Actin filaments on myosin beds: The velocity distribution. Physical Review. E. Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics 52(6):6573-6579.

Ishijima A, Harada Y, Kojima H, Funatsu T, Higuchi H, Yanagida T. 1994. Single-molecule analysis of the actomyosin motor using nano-manipulation. Biochem Biophys Res Commun 199(2):1057-63.

Kinose F, Wang SX, Kidambi US, Moncman CL, Winkelmann DA. 1996. Glycine 699 is pivotal for the motor activity of skeletal muscle myosin. J Cell Biol 134(4):895-909.

Kron SJ, Spudich JA. 1986. Fluorescent actin filaments move on myosin fixed to a glass surface. Proc Natl Acad Sci U S A 83(17):6272-6.

Sellers JR, Cuda G, Wang F, Homsher E. 1993. Myosin-specific adaptations of the motility assay. Methods Cell Biol 39:23-49.

Uyeda TQ, Kron SJ, Spudich JA. 1990. Myosin step size. Estimation from slow sliding movement of actin over low densities of heavy meromyosin. J Mol Biol 214(3):699-710.

Winkelmann DA, Bourdieu L, Ott A, Kinose F, Libchaber A. 1995. Flexibility of myosin attachment to surfaces influences F-actin motion. Biophys J 68(6):2444-53.

Video Capture

We have used a Scion® LG3 capture board in an 866 mHz Apple Macintosh® G4 running Mac OS 9.2 to capture video either directly from the microscope or from SVHS video tape. Scanalytics® IpLab image analysis software is used to control the LG3 board and capture digital video as an image stack. This combination is capable of capturing full frame video (640 x 480 pixels) at 30 frames per sec. Routinely, frames rates of 10-15 frame per second are suitable for analysis. A dataset corresponds to 30 - 60 sec or longer of continuous digitized video. One minute of video collected at 15 fps produces a 900 frame stack. If each image is 600 x 460 pixel and 8 bits deep (byte format) this image stack corresponds to a 236.9 MB file. The IpLab *live LG3 capture* module digitizes time lapse video directly to RAM and thus, requires a large amount of physical memory committed to the program: at least as much as the size of the digital file you want to collect.

The LG3 live capture module of IpLab has an option to collect a timing window that saves in a separate file 'time stamps' (from the system clock) for each frame collected. This information is invaluable in establishing the time base for the collected video, and its use is highly recommended. We have found that on a freshly booted Mac G4 running Mac OS 9.2 you can collect full frame video at a rate of 30 fps. However, if the machine has been running for several days, the rate of data collection is often less than 10 fps independent of what has been set in the live capture setup window. I have experienced this problem on several occasions and cannot explain it. The time base for motion analysis is too important to leave to chance, so always collect a timing window, and check the rate of data collection after saving the file.

The continued development of the Mac OSX operating system and it superiority over the older operating system has made it increasingly difficult to maintain and older OS 9.x computer. Scanalytics has developed an OSX version of IpLab (version 3.9 and up) that is vastly superior to the OS 9.x version, but is no longer supports the Scion LG3 board. This forced the development of an alternative method of image capture. We have used ImageJ, the Java implementation of NIH Image from Wayne Rasband along with a fast Scion grabber board pluggin for ImageJ: JNIAkiz to acquire full frame rate images at 30 fps. This pluggin was developed by Adrian Daerr and is available at http://www.pmmh.espci.fr/~daerr/progs.html#ScionFGAkiz. ImageJ (1.32) capture the data as 8-bit Tiff image stacks of about the same size as the IpLab image stacks noted above, so the same demands on System resources exist for this acquisition scheme.

The best data for analysis is collected directly from the video stream of the imaging system. Alternatively, the dataset can be collected from SVHS videotape after the completion of the experiment. Direct capture assures the highest resolution in the dataset, since even SVHS VCR recording results in considerable loss of image resolution. However, videotape is often more convenient and some loss is tolerable in all but the most extreme cases of poor video.

Data File Types

Starting with version 1.4 of the Tracking programs Scanalytics IpLab 8-bit image stacks up to 640 x 480 pixel/frame and an 8-bit Tiff image stacks of the same frame size are supported. The addition of the Tiff image stack to the acceptable data types provides a standard data format that should be widely available with different image acquisition programs. There are really only two tracking programs that need to access the original video files (setfilters and grabworms), so development of the tracking programs for other image file types is possible. All that is needed are the details of the data structure, some experience with *C* programming, and a suitable development environment. Source code for all of the tracking programs is available (e.g. see Appendix C).

The Motility Analysis Programs

The analysis of the image stacks involves processing by a series of programs developed specifically for this purpose. This version of the software has been developed with the free Apple development environment, *Project Builder,* to run under Mac OSX 10.2 and higher (current version 10.3.7). All of the code is written in standard *C* and many of the programs require no graphics, so they are directly portable to any Unix or Linux platform. The graphics programs use *OpenGL* and much of that code is portable as well. The original versions of the analysis programs were written in the old *gl* language for an SGI workstation running IRIX 4. The porting from *gl* to *OpenGL* was started on a Windows development platform and much of the code has been tested on Linux platforms as well. The current implementation of the software should easily port to other platforms. However, cross-platform porting is not the focus of this document. That is to say, that I've tested the programs on linux and Mac OSX, but I do not have a Windows development platform and leave that extension to anyone willing to give it a go.

The individual tracking programs are listed below in the order that they are normally used and briefly described. The remainder of this document provides a detailed description of what each program does and how it is used to analyze motility assay video. Appendix C of this document contains the *C* programming language source code of all of the programs described and a tar archive of the source code is available on the distribution CD. A sample data set also is provided on the CD to demonstrate just how these programs handle the video data.

The Tracking Programs:

- ***setfilters*** – An *OpenGL* program that sets the parameters for threshold and sharpening filters used to separate the filaments from the background.
- ***graworms*** – A command line program used to apply the threshold and sharpening filters to each frame, and extract the coordinates of every filament in every frame (i.e. make the cwb files).
- ***analyze*** – This is a command script that invokes a sequence of programs (*threadworms*, *untangle*, *makewormlist*, and *asciibatchns*). These programs sequentially analyze the movement of individual filaments frame to frame, define the individual tracks, and extract the position coordinates and time data.
- ***showcwb*** – An *OpenGL* animation program that displays the filaments extracted from the video sequence.
- ***showswb*** – An *OpenGL* animation program that displays the individual filament tracks and motion extracted from the video sequence.
- ***speed*** – An *OpenGL* program that displays the distance vs. time data, edits the data and records the speeds, filament lengths, movement distances and duration of the individual filaments. It also has an internal animation program that can be used to display the motion of the filament being edited.
- And finally, ***DataTemplate.xlt***, an ***Excel*** spreadsheet *template* that can be used to interpret the velocity data.

***Setfilters*: Setting the Threshold and Filter Parameters file**

It should be noted that these programs were initially developed on a SGI workstation with a three-button mouse and were designed to use that 3-button feature. However, Mac mice come with a single button, so, if you are working an a Mac OSX computer the purchase of a three-button mouse is a big plus (e.g. a Kensinton StudioMouse). If you don't have a three button mouse or are working on a laptop, the button definitions are as follows: the **Left** button is a simple click (of the mouse button), the **Middle** button is *option key plus click* combination (***option-click***) and the **Right** button is the *control key plus click* combination (***cntrl-click***).

The first step in analyzing the collected data is to define the parameters required to separate the foreground actin filaments from the background noise. These parameters are stored in a file named '.Parameter' that is created by the interactive *OpenGL* graphics program, *setfilters*. The parameters include a threshhold algorithm for setting a *cut-off* value for the foreground filaments and a filter kernel for sharpening the filament outlines. This program is run from command line in a *Terminal* shell from the directory that contains the image stack by issuing the command: '*setfilters filename*' (where '*filename*' is the image stack in either IpLab or Tiff file format). An *OpenGL* graphics window will open in front of the *Terminal* window. Move the pointer over the red text box in the window. To display an image, place the cursor over the 'Show Unfiltered' text and press the **middle** button twice to display the image (*option-click* = ***middle*** *mouse button*). An image from the stack will appear as a color image with the color map defined by the histogram just to the left of the red text window (see Figure 1). This program creates a '.Parameters' file is none exists in the directory or it opens and modifies and existing file.

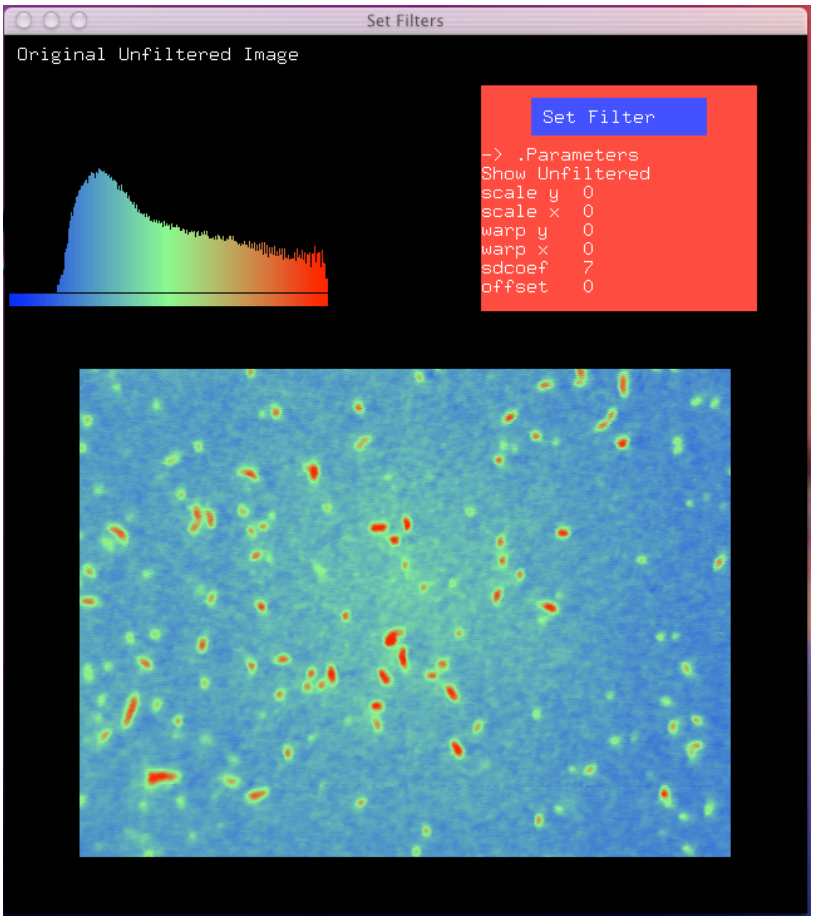

Figure 1. The *Set Filters* window showing the unfiltered first image of an image stack. The text lines with parameter values in the red box are *active lines*. ***Right* or *Left*** mouse clicks while the pointer is positioned over these lines increments and decrements the parameters values. A ***Middle*** mouse click applies the threshold values to the image in the window.

Right* mouse button = *ctrl+click

Middle* mouse button = *option+click

Left* mouse button = *click

The filaments appear in red, outlined in cyan and surrounded by a pool of blue-green noise. The parameters that will be used to filter the image are listed in the red text box. Each of the white lines in the text box is an *active line*: the parameter value on the line can be changed with *key-click* combinations. The line value can be increased with the ***right*** mouse button (***ctrl-click*** combination) when the pointer arrow is positioned over the line of interest. The ***left*** mouse button (a simple ***click***) decreases the value of the line that is beneath the pointer. The ***middle*** mouse button *(**option-click*** combination) on any of the active parameter lines applies all the current parameters values to the image in the window. Thus, the effect on the active image of incrementing and decrementing the values in the text window can be immediately displayed. Using this program, you can interactively modify the *thresholding algorithm* and observe the effect on the objects in the window. The unfiltered original window is redisplayed with a ***middle*** mouse button click (***option-click***) on the 'Show Unfiltered' line. The values are saved to a '.Parameters' file by a ***middle*** mouse button click (***option-click***) on the '–> .Parameters' line. You can quit *setfilters* at any time with the '**q**' keystroke.

Setting the Cut-off Threshold

Setfilters allows the user to interactively define the best settings for separating the foreground filaments from the background noise. The threshold for separating the filaments is set by the 'sdcoef' and 'offset' parameters. The *sdcoef* is the number of standard deviations (x 10) above the *mode* of the intensity histogram that will be used as the cutoff for separating the foreground from the background. The *offset* is an additional value added to the threshold. The *cutoff* parameter is defined by the equation:

$$cutoff = mode + (sdcoef/10 \times SD) + offset.$$

The *mode* is the intensity value (0-255 range) of the peak in the intensity distribution histogram. The standard deviation (SD) of the pixel intensity distribution of the image is used to define how much above the *mode* the actin filament intensities are distributed. Generally a *sdcoef* of ~14-22 (1.4-2.2 x SD) and an *offset* of 8-10 are used to set the *cutoff*. Values below the *cutoff* are set to zero and values above the *cutoff* are stretched to a range of 0-63.

Frequently, video cameras have a non-linear response across the field of view especially when "pushed" to increase camera sensitivity for the motility assay. This gives rise to a bright area in the center of the image and a fall-off in intensity towards the edges. The 'scale y', 'scale x' and 'warp y', 'warp x' parameters allows the user to correct for both foreground and background intensity fall-off, providing a means to flatten the image before processing. The *warp x* and *warp y* parameters are manipulated first and these parameters change the scaling of the background intensity in x and y respectively. Increasing these parameters increases the scaling of the pixel values near the edge using quadratic weighting based on the distance from the center of the image. If the *sdcoef* is set to a low value (e.g. 4), increasing the '*warp*' factors will cause a brightening of the background near the edges (remember to use the **Middle** button (***option-click***) over any active line to update the image once you change a value to see the effect of the change). The goal is to have a uniformly flat background. *Warp* values of 3-8 are typical with our vidicon camera (a DAGE 70 newvicon video camera with a Hamamatsu image intensifier). Our CCD camera is less sensitive and noisier than our vidicon but it has a flatter background, so it requires lower *warp* values than the vidicon.

The 'scale x, scale y' parameters increase the intensity of the foreground objects. When the background is scaled up with the *warp* factors the foreground objects near the edge can get lost in the noise. The *scale* factors increase the intensity of the foreground objects again using quadratic weighting based on the distance from the center. Typical *scale* factors of 4-8 are used with our vidicon camera. (Remember again to update the **Middle** button (***option-click***) over any of the *active lines* to see the effect of changes in the foreground object intensity).

To begin selecting parameters for a new dataset, use starting values of 'sdcoef = 4' and 'offset = 0' and then set the background and foreground intensities with the 'warp' and 'scale' factors, respectively. The *sdcoef* and *offset* can then be adjusted to achieve the desired *cutoff*. An example of properly set parameters is shown in the Figure 2.

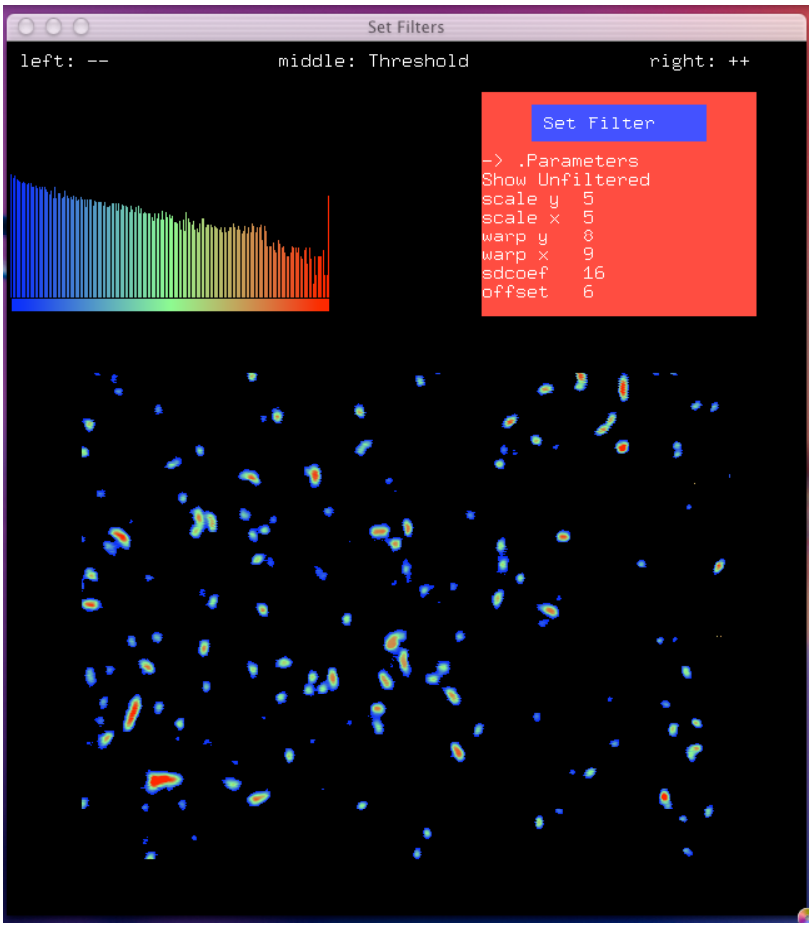

Figure 2. The same image as in Fig. 1 with the set of parameters on the active lines applied to the image. Most of the filaments are now easily separated from the background noise. Note the changes in the histogram. Only intensities that exceed the *cutoff* are displayed above the black background.

Setting the Sharpening Filter Kernel

Near the top of the red text box there is a blue button that is labeled **Set Filter**. This button actually toggles between **Set Filter** and **Set Threshold**. When you are setting the threshold, the blue button reads **Set Filter**. *Clicking* on the button toggles the red text window to a different menu structure that provides access to the parameters for setting the sharpening filter kernel. A white **Set Threshold** button that is used to toggle back to the threshold parameter menu replaces the blue **Set Filter** button. With any new dataset, start with the threshold parameters and then set the filter parameters. However, one can go back and forth between the two menus and make changes to the threshold or filters at anytime and experience will help define when that is advantageous.

The Set Filters menu is used to work on the refocusing filter parameters: *expand*, *clipHi*, and *clipLow*. *Expand* scales and stretches the intensity range of the filaments. Increasing it results in brighter and thicker filaments. This scaling helps reduce fragmentation of the filaments as the intensity drops near the edges of the image. *ClipHi* and *ClipLow* define the edge profile of the filaments: values of *clipHi* = 15-20 and *clipLow* = 6-10 are typical. When these values are applied after thresholding most of the remaining noise disappears. The final objects that will be extracted from the video frame can be viewed by a simple *click* (***Left*** mouse button) on 'Threshold and Separate' line (Figure 3). If you are satisfied with what you see, the set of parameters can be saved with ***Middle*** mouse button (*option–clicking*) on '-> .Parameters'. A message across the top of the window will indicate when saving the '.Parameters' file is complete, and you can exit *setfilters* by pressing '**q**' on the keyboard.

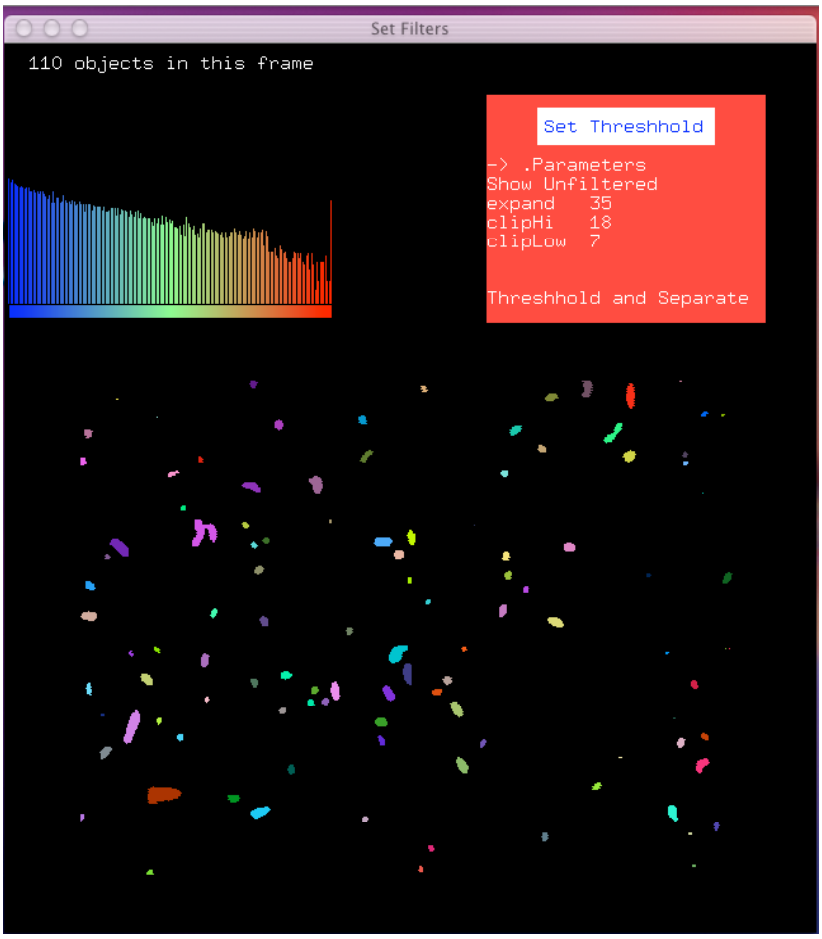

Figure 3. Objects in scene after activation the 'Threshhold and Separate' button. This is the same image as in the previous examples. Note that 110 objects have been counted in this scene. This includes some small objects that will not be extracted with *grabworms*. There is currently a size threshold of 15 pixel^2 for the smallest object that will be saved by *grabworms*. With our imaging setup that would correspond to a 0.8 x 0.4 μm filament. These really small filaments flicker in and out of view and are dificult to measure. This threshold can be changed if so desired.

***grabworms*: Capturing the "*worm*" boundaries**

Once you have created a '.Parameters' file for a specific image stack, it is time to extract the individual filament boundary coordinates (or *worms*) from each frame of the video. This is done with the program *grabworms* that is started from a *Terminal* shell with the command: '*grabworms filename*' (where '*filename*' is a byte image stack containing the video sequence of interest). The *grabworms* program opens and reads the '.Parameters' file created with *setfilters* and applies the filters to each frame of the image stack to separated the foreground filaments (*worms*) from the background and extract the boundary coordinates for every worm in each frame. It creates a group of *cwb* files (captured worm boundaries). Each *cwb* file contains a binary list of coordinates and frame reference for every *worm* in 100 consecutive frames. Because the *cwb* files correspond to video increments of 100 frames each, the minimum number of frames that can be analyzed is 100. There is no maximum number of frames, but frames are stored in increments of 100. So an image stack with 499 frames with produce 4 *cwb* files (Not 5 *cwb* files! *Iplworms* will ignore the last 99 frames!) designated: cwb.00000 cwb.00100, cwb.00200 and cwb.00300. This is obviously a consideration when collecting the original video files. Always collect in increments of 100 frames.

Grabworms has no graphics component and considering what it does, it is fairly fast; however, it still does take several minutes to capture the worm boundaries for a 900 frame 640 x 480 pixel image stack. The program reports to the *Terminal* window the frame number and the number of object in the frame that have been saved. Once extracted from the video files, the *cwb* files can be saved and analyzed later or passed directly on to the *analyze* script.

The *analyze* Script

The analysis of the *cwb* files is easily done with the command script *analyze*, which is found in /usr/local/bin and is on the command path of most user shells. From a *Terminal* shell in a directory containing a group of *cwb* files start the analysis with the command: '*analyze*'. The script that executes with this command is quite simple and is listed below.

analyze

```
echo First we thread the worms in time:
threadworms cwb*
echo Then we untangle the braid:
untangle
echo Then we extract all worms with a timeline longer than 20 time steps
makewormlist 20 | sort -r | csh
echo Now we generate tracks (trk) and ascii (asc) files:
asciibatchns swb*
```

The Individual Analysis Programs

***threadworms and untangle:* Define the links between objects**

The *cwb* files are first analyzed by *threadworms*. This program links the individual objects in each frame of the *cwb* files. The output file produced by *threadworms* is *thread,* and it contains a binary list of all the objects (*worms*) linked through space and time. Some of the threads will naturally be intersecting as objects collide or cross each other. This happens quite frequently in typical assay videos. These intersections must be sorted out by the next program, *untangle*. This program sorts the braided and twisted threads and

inserts breaks in threads where objects cross one another. The output of *untangle* is a *braid* file. Unlike untangling a ball of string, the *untangle* program cannot predict where each filament goes after a collision and, therefore, cannot resume the appropriate thread for each object of the collision. Rather, the 'threads' are discontinued at the collision and new threads are started for each object that emerges from the collision. Thus, to obtain long and uninterrupted threads, it is best to optimize the actin filament concentration in the flow cell.

showcwb*: Animate the Captured *worms

After threading and braiding, the *cwb* files together with the *braid* file can be displayed using the *OpenGL* graphics program 'showcwb'. This is run from a *Terminal* shell in the directory containing the *cwb* and *braid* files with the command: '*showcwb cwb**'. An *OpenGL* graphics window will open and a click inside the window will reveal a menu to run or quit the animation. The set of *cwb* files with be read in order and multicolored objects displayed. The color assignment is random, however, as long as a thread is maintained for any individual object, that object will remain a single color. The colors change when the thread is broken, either by a collision or a flicker due to a frame loss for small and dim objects. The program currently runs through all the *cwb* files then stops, a mouse click will reveal the run/quit menu again and you can repeat or quit the animation. You can also exit the program with the '**q**' keystroke. The animation runs faster than real-time and provides an excellent display of the captured *worms*.

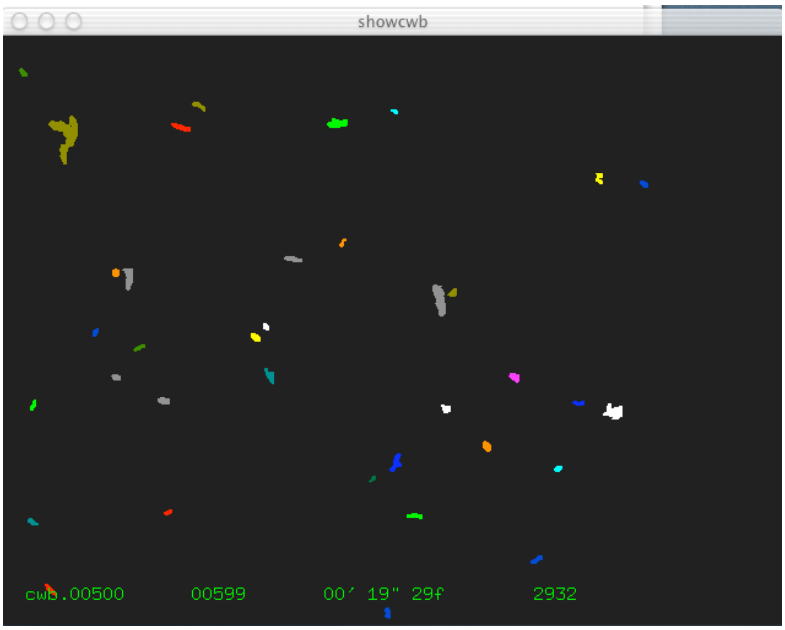

Figure 4. The showcwb display used to animate the captured cwb files.

Analyze individual objects: *makewormlist and asciibatchns*

Extracting individual tracks from the braided list is done with two programs: *makewormlist* and *asciibatchns*. This is done automatically with the *analyze* script. The program *makewormlist* creates the *swb* output files (***s**ingle-**w**orm-**b**oundaries*). There will be one *swb* file for each object that has been braided from the *cwb* files. This program takes an argument that sets the minimum duration of an *swb* file. The default is 20 *ticks*. Each frame is considered a *tick* so if the data were collected at 30 fps, then 20 ticks would set a minimum duration of 0.67 sec for a valid *swb* file. The *tick* limit is used to delete the *swb* files of stationary filaments. If you want the stationary objects included in the analysis, then set this value to '0' in the *analyze* script; however, this will significantly increase the number of files that are generated and analyzed.

The *swb* files generated by *makewormlist* are analyzed by *asciibatchns* creating the *trk* and *asc* files. Each *trk* file defines the track produced by a single worm (*swb*). The *trk* file is the skeletonized superposition of the object in all the frames included in the *swb* file and represents the track followed by the object. The *asc* files are the coordinates of the centroid of the object projected onto the track for each *tick* (frame), and the coordinates of the front tip and back tip of the object. There is one *trk* and one *asc* file for each *swb* file. The *asc* file is a standard ASCII text that can be exported and plotted if so desired, but the *speed* editor does exactly this, so why bother. The *speed* editor also provides a method to edit and insert breakpoints in the speed plots and determines the slope of the distance vs. time data: the filament velocity.

The program *asciibatchns* reads all the *swb* files created by *makewormlist*, but it cannot interpret all *swb* files. As a result many *swb* files are deleted at this stage because the topography of the track is too complex for interpretation by the current algorithms, or the length of the track is too short.

showswb: Animate individual objects moving along the tracks

The *swb* and *trk* files can be displayed with the *OpenGL* graphics program *showswb*. Like *showcwb*, this program is executed from a *Terminal* shell with the command: '*showswb swb**'.

Figure 5. The *showswb* window showing the filament and track of one of the *swb* files. Note that this is file number 50 of 107 in this data set. This OpenGL animation window can be used to review the results of the analyze programs.

The *showswb* graphics window will appear behind the *Terminal* window. Drag it to the front and read the display instructions. Keyboard commands are used to display the next (**n**) filament, the previous (**p**) filament, redisplay (**r**) the last filament, and quit (**q**) the program. *Click* in the graphics window to start the first track. A red line corresponding to the track followed by the filament is drawn and a white object with a blue bounding box moves along the track. This is the digital translation of the motion of individual filaments extracted from the original video. The filename of the current filament and the number of individual filaments in the dataset are displayed in the upper left corner of the window. You can look at all the filaments, one at a time, or quit at any time. Note that the speed of the filaments is accelerated to facilitate the viewing a complete set of *swb* files in a reasonable amount of time.

The *Speed Editor*: An Interactive Editor of Filament Motion

The final step in the analysis is the *speed* editor. The *speed* editor is an interactive *OpenGL* program that analyzes the distance vs. time data for individual swb files and outputs an ASCII *data* file that contains the duration of movement, filament speed, and filament length. The *speed* editor is run from a *Terminal* shell with the command: '*speed asc**'. An *OpenGL* graphics window will appear; click on the orange button labeled **First File** and the distance vs. time plot of the first *asc* file in the data set will be displayed. The original data is plotted with a white line, the initial fit of the data is plotted with a red line and a smoothed fit of the data is plotted with a yellow line. If the filament moved with a constant velocity, the initial fit is a straight line and it's the only fit that you want. You can move to the next filament by clicking on the **Next File** button. However, if the filament stalled or moved at several different velocities, the fit can be segmented to reflect these different speeds by using the mouse to interactively insert breaks in the fit. To insert a break, position the pointer at the location of the desired breakpoint and press the ***Middle*** mouse button (*option-click*). This will insert a break in the red line and create two segments reflecting the different slopes. The break can be removed by repositioning the pointer on the breakpoint and then simply *clicking* the mouse button (***Left*** mouse button).

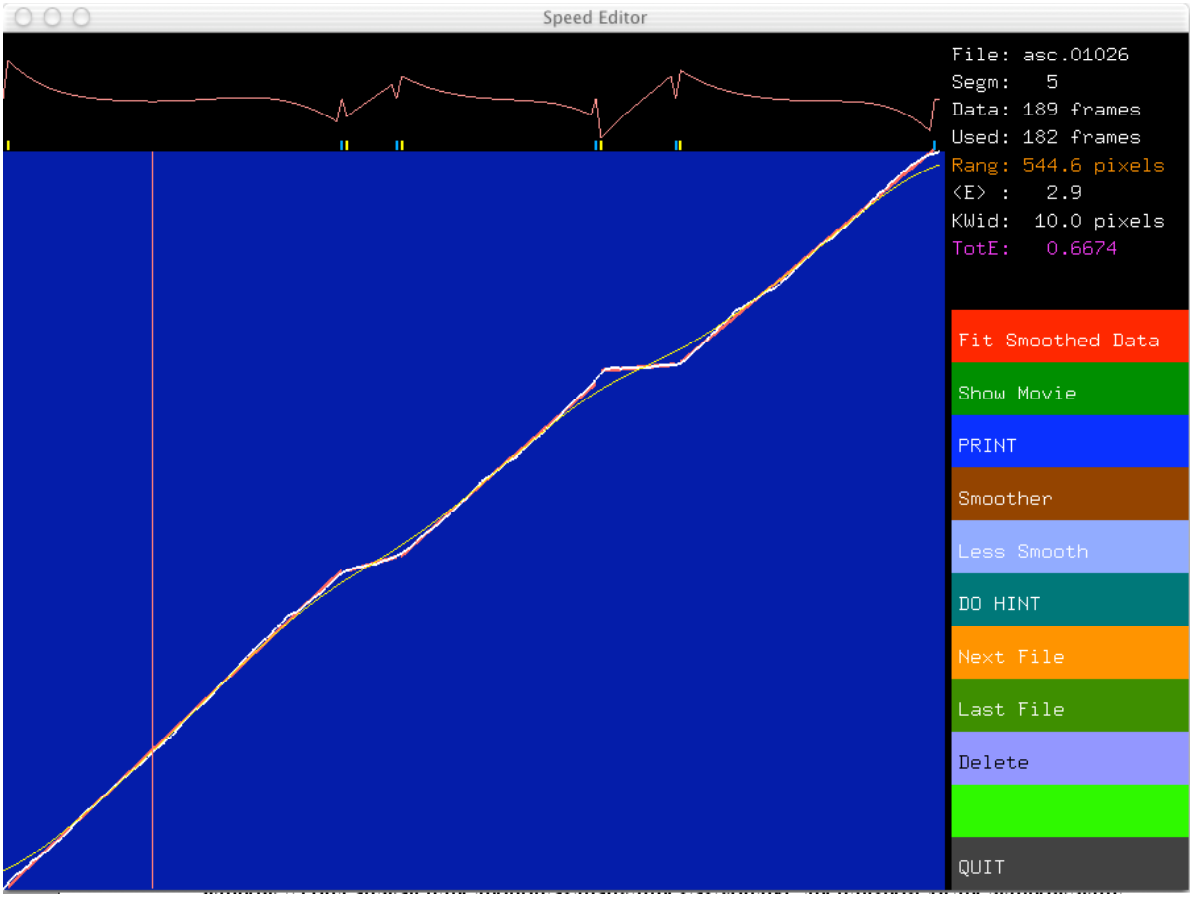

Figure 6. The *Speed Editor* window.showing the distance vs. time plot for file, asc.01026. The fit to the original data has already been broken into 5 line segment. A vertical blue line marks the position of another possible breakpoint that can be taken by *clicking* on the 'DO HINT' button from the menu on the right.

The window running across the top, above the data plot, shows a residual plot of the difference between the fit and actual data. At the lower margin of that residual window are yellow (left) and blue (right) hash marks that show the limits of the segments in the fit. Any number of breakpoints can be edited into the

data to achieve the best representation of the actual speeds that the filament moved including any stalls that it made during that movement.

Displayed in the upper right-hand corner of the *Speed Editor* window is a listing of: the current filename, the number of segment in the data fit, the total number of frames in the plot, how many of those frames are used in the fit, the range (distance in pixels) that the filament moved, the width of the smoothing kernel (Kwid) and a total error factor. Moving down from this information box, there are a series of buttons that operate on the data in the graphics window. To some extent, these buttons are self-explanatory.

These buttons are actuated with a simple *click* (***Left*** mouse button):

Fit Smoothed Data/Fit Original Data (A toggle button)
Show Movie (Display an animation of the file being edited)
PRINT (Send a postscript copy of this window to a network printer - lpr)
Smoother (Decrease the KWID value)
Less Smooth (Increase the KWID value)
Do Hint (Make a break at the suggested position-vertical cyan line)
Next File (Move on to the next file in the dataset)
Last File (Go back one file and recheck the segmentation)
Delete (Delete the current asc*file. Used when a track is not interpretable.)
QUIT (Get me out of here and save the *data* file)

Obviously, you move to the next file with the **Next File** button or go back to the previous file with the **Last File** button and quit with the **QUIT** button. Often a cyan line drops down at a point along the plot where the fit deviates from the data. The **Do Hint** button can be used to create a break at this point. For many files, taking the **Do Hint** option is all you need and it's a fast way to edit the dataset. The keyboard shortcut - h (for hint) is available for **Do Hint** to make this choice even faster! However, in some cases that choice is not the best and manual insertion of breaks may be preferred. Remember, point at the breakpoint and use the ***Middle*** mouse button (*option-click*) to insert the break.

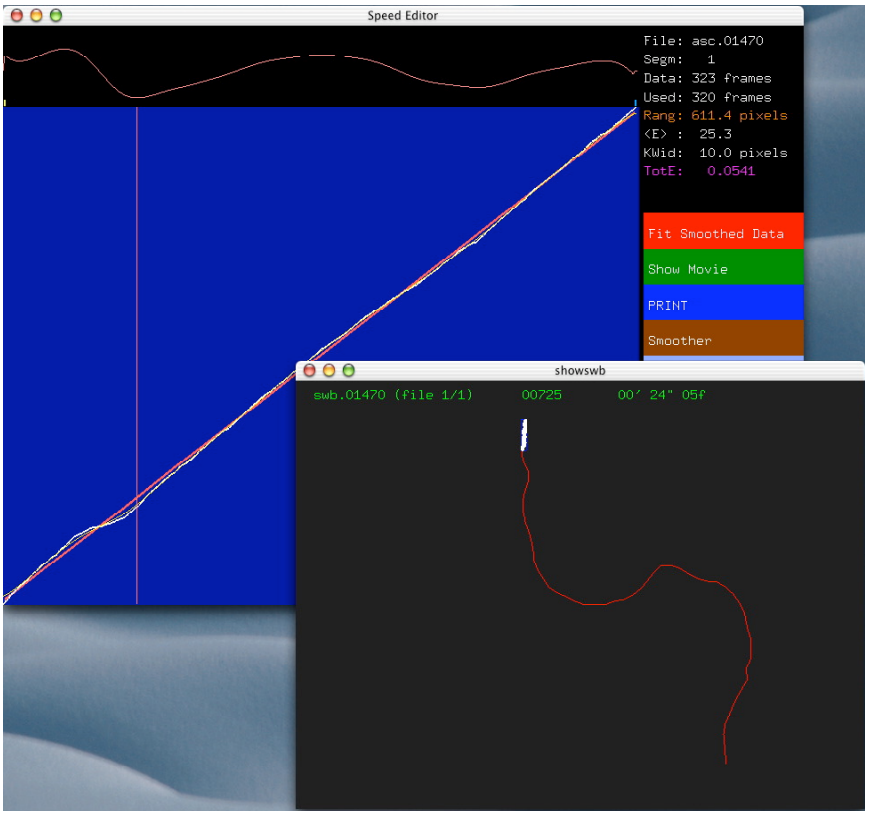

Figure 7. The *showswb* window appears when you click on the **Show Movie** button so you can review the filament motion that corrresponds to the *Speed Editor* graphic. Clicking in the *showswb* window starts and repeats the movie. A '**q**' keystroke closes the window.

Sometimes it is difficult to know just what the line plot represents. The **Show Movie** button pops-up a *showswb* window shows the track and movement of the filament that is plotted in the editing window. You can watch the motion as many times as you like.

When you have finished segmenting the distance vs. time plots for all the *.asc files, the **Next File** button changes to **That's All!** and the analysis is complete. The *speed* editor creates a file named '*data*' or opens and existing data file. This file contains a formatted listing of the segmented data. The file lists the start and stop frame number of the line segments, the speed in pixels/frame, the length of the filament and the *asc* filename identifier. The *data* file is saved whenever a new filament plot is displayed. So, you can **QUIT** the *speed* editor at anytime after you start an analysis. You can always return to the last file you were editing and resume the analysis where you left off.

When you have finished analyzing the asc files it is recommended that you **MOVE** the *data* file to a new filename with the Unix command: mv *data* '*newfilename.data*'. This will save the data and prevent you from overwriting it when you analyze another set of *asc* files. Never analyze two different data sets using the same *data* file. Remove the old file by renaming it or deleting it before you start a new analysis. At this stage the data in the file can be analyzed, plotted and interpreted. To assist in this, an excel spreadsheet template is provided.

***DataTemplate.xlt*: An Excel template for analysis of the *data* file**

A complete analysis of 1 min of video will usually contain hundreds of movement segments. The formatted ASCII *data* file is readily imported into commercial spreadsheet software to complete the analysis.

The *data* file contains the entries that are tabulated in the format shown here:

From Frame	To Frame	Speed (pixels/frame)	Speed Error	Length (pixels)	Length Error	asc.* Filename
0000	0000	000.00000	00.00000	000.0000	00.00000	asc.000000

An Excel® template named *DataTemplate.xlt* is included with the distribution to aid in interpretation of the *data* file. Since the *data* file tabulates the results in pixels and frames, one of the first steps is converting these values to micrometers and seconds. This requires knowledge of the raster of the original data (μm/pixel) and the time base (Frames Per Second: FPS). Cells are provided in *DataTemplate.xlt* to enter this information and convert the *data* file to useful numbers.

To use *DataTemplate.xlt* open the file in Excel and enable the macros that are embedded in the template. Next open the *data* file as a separate spreadsheet and follow the import prompts. When the data file is open, you may want to examine it for lines that say 'NAN'. These are errors in the segmentation that need to be deleted. Delete them now. Next do an ascending sort of all the data on the basis of the filament speed (column C) and delete those speeds that are negative. Negative velocities generally do not make any sense. They arise because sometime the program does not know if a filament is coming or going. Now copy the 7 columns of information from the data file and paste them in columns A3 through G3 of the *DataTemplate* file. Next select cells H3:K3 and copy the contents of these cells down to the last row of the data you just imported into columns A through G. Now use the key combination *option+command+s* and the data will be sorted and analyzed automatically.

At the top of the *DataTemplate* file, in columns L through Y there is a Summary of the dataset, a Length Distribution Analysis, and a Speed Distribution Analysis. This summary data can be graphically analyzed either within excel or in other programs such as *Kaleidagraph*®. The formulas used for the calculations in the spreadsheet are quite complex, but they are all thoroughly described in the Excel Help and can be easily examined in the spreadsheet. When you have finished importing data, rename the worksheet and save the file under an appropriate filename. Note that the template file has multiple worksheets and can be used to analyze multiple datasets, providing a convenient way to group results from a single experiment.

***Cleanup_tracking*: Cleaning-up when you're finished**

Finally, when you have finished with an analysis what is it that needs to be saved? I usually create a tape archive of the *cwb* files generated from an image stack and the final analyzed *data* file. The rest of the analysis can be repeated with the *analyze* script if you've kept the *cwb* files. You can display of the "original" video with the animation program *showcwb* if you have saved the *cwb* and *braid* files. To archive the files use the Unix tar command:

tar cvf your_filename.tar cwb* thread braid *newfilename.data*

Once this is done, the asc*, braid, cwb*, swb*, thread, and trk* files that remain in the directory can be discarded with the command script: *cleanup_tracking*. This script is included in the binary distribution. It simply deletes all the files associated with an analysis. So be careful, backup what you want to save before you evoke *cleanup_tracking*. It asks no questions, and it takes no time at all to do its thing! The script does not delete the *data* file created by the *Speed Editor,* so if you forget to rename this file, all is not lost. However, be careful to remove or rename this file before you start the analysis of a new set of *asc* files or the *Speed Editor* will attempt to use the old *data* file for the new analysis. That gives rise to some very odd segmentation results.

Appendix A: Installation Notes

A CD with the binary files, source code, an analyzed test data set, an image stack for testing the programs, this user Manual and an Excel template file has been provided. Eventually, all of these files will be available as a download on my WEB site packed in a single (and large) zipped tape archive (tar.gz) file.

The Contents of the CD:

```
-rwxrwxrwx   1 daw  99      374272 Apr 28 20:33 DataTemplate.xlt
-rw-r--r--   1 daw  99      997376 May  7 19:47 Manual.doc
-rwxrwxrwx   1 daw  99      374767 May  8 15:39 Manual.pdf
drwxr-xr-x   5 daw  99         264 May  7 19:53 test_ImageStack
-rw-r--r--   1 daw  staff  4997120 May  7 19:46 test_analyzed.tar
-rw-r--r--   1 daw  staff   307200 May  7 19:46 tracking_bin.tar
-rw-r--r--   1 daw  staff  2795520 May  7 19:49 tracking_src.tar
```

Files with the '.tar' extension are tape archive files. The test_ImageStack is a directory that contains a sample IpLab image stack (600 x 460 pixels, 900 frames) with a sample '.Parameters' file. Manual.doc is a Word® file of this documentation and *DataTemplate.xlt* is an Excel® template file for analyzing the final *data* file produced by the *Speed Editor*.

Installation is done on a computer running Mac OS X, and it does requires some fundamental Unix skills and an Mac OS X administrator password. It is easily done from a *Terminal* shell, so open one now. First insert the CD and let it mount. Next, you will copy the tape archive files to appropriate directories and extract the contents using the commands listed below. For most multi-user Unix machines, a general /usr/local/bin directory is maintained for locally developed applications. This is the appropriate directory for these programs and will be used for installation of the binary files. The source code for the programs should be place in a directory owned by a local programmer or system maintainer. I use the Library directory in my local directory structure. Similarly, the test data should be put in a directory owned by a user. I will use my username (daw) and directory structure as an example for the installation. In the following example the *Terminal* prompt is indicated by "**[altair] daw%**" and the text following the prompt should be entered by you.

The CD is mounted at /Volumes/Tracking_CD and you are working in a *Terminal* shell in your home directory - proceed>

[altair] daw% cd /usr/local/
[altair] daw% sudo tar xvf /Volumes/Tracking_CD/tracking_bin.tar
password: (enter the administrator password at the prompt then <return>)
{At this point the programs will be extracted and place in /usr/local/bin}
[altair] daw% cd ~/Library/
[altair] daw% mkdir Tracking
[altair] daw% cd Tracking
[altair] daw% tar xvf /Volumes/Tracking_CD/tracking_src.tar
{The source code will be extracted and placed in ~/Library/Tracking/}
[altair] daw% tar xvf /Volumes/Tracking_CD/test_analyzes.tar
{The test data will be extracted and place in ~/Library/Tracking/test_analyzed/}
[altair] daw% cp /Volumes/Tracking_CD/test_ImageStack . (don't leave out the period or you'll be told you didn't specify a destination!)
{The directory will be copied to ~/Library/Tracking/. This is a large directory and the transfer from CD will take time so be patient.}

[altair] daw% cp /Volumes/Tracking_CD/Manual.doc ~/Documents/.
[altair] daw% cp /Volumes/Tracking_CD/DataTemplate.xlt ~/Documents/.

So, now we have the contents of the CD transferred to disk and you are almost ready to use the programs, but not quite. First you will need to close the *Terminal* window you are now in and open a new one. This is because the new files you loaded in /usr/local/bin are not "known" to this *Terminal* window but will be when you open a new one (It's a unix shell thing!). You may need to make sure that /usr/local/bin is on your path to use the programs.

There is a Makefile in the src directory that will build the set of programs from source. Edit the Makefile and redefine the bin directory or designate a different compiler as necessary for your installation. Invode the command ***make*** in the src directory and it will make a clean set of executables. The command ***make clean*** will clean up the object files when the build is complete. The code is all there and available to modifiy and rebuild.

You will noticed that there are alternative definition lines in the Makefile for compiling the code on Linux machines. A big differences between Linux X86 and Mac OSX is the definition of the GL libraries and header files. This has been handled in the Makefile. Another major problem is the big-endian/little-endian difference between the Motorolla and Intel chips. All of the code does run on a Linux X86 machine we tested here; however, the *setfilters* and *iplworms* programs open the original image files and work is needed to define alternative file structures for Linux and Windows systems. I have not invested the time to define a new file structure for these other operating systems. I have kluged the read_ipldataset.c program for the Linux distribution so that the Iplab test data can be read and processed. This is a temporary fix at best.

From here on you are on your own. Good luck and my email address is: daw@pleiad.umdnj.edu.

Appendix B: Known bugs

There are probably more than I've found so far, because I know what the programs can't do, so I never have tried to crash them in every possible way that a new user will try... But here goes a first effort.

setfilters:

None so far. Its has one failing in that it only opens IpLab or Tiff image stacks of maximum size 640 x 480 and in byte format. It runs quite well even with huge image stacks, but is a little slow opening really big files.

iplworms:

None so far. Like *setfilters*, *iplworms* is limited to IpLab or Tiff image stacks of maximum size 640 x 480 and in byte format.

analyze script:

Problem: The program *makewormlist* fails if you have left too much noise in the images when you set the threshold and filter parameters. The noise is mistaken for objects that appear and disappear and it really mucks up the sorting and creation of the *swb* files. You know you have this problem if there are many more *swb* files than *trk* and *asc* files and you don't get a "Bye, Bye!" at the end of running the analyze script.

Solution: Reset the '.Parameters' file with *setfilters* and don't be so greedy! Then rerun the analysis and it should go just fine.

showcwb:

The cwb files can't be displayed until you have the thread and braid files created with the analyze script.

showswb:

None so far.

speed editor:

Problems: **Print** only prints to a network printer and I'm not sure why the code even works, but it does. The code is quite old and in a postscript language I don't remember anymore. I'm just happy I got the print button to work at all.