SANDIA REPORT
SAND2014-1632
Unlimited Release
Printed February, 2014

LineVISAR: A Fringe-Trace Data Analysis Program

Michael D. Furnish
Dynamic Materials Department (1646)

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2014-1632
Unlimited Release
Printed February, 2014

LineVISAR: A Fringe-Trace Data Analysis Program

Michael D. Furnish
Dynamic Material Department
P.O. Box 5800
Albuquerque NM 87185-1195

Abstract

The line-imaging ORVIS or VISAR provides velocity as a function of position and time for a line on an experimental setup via a streak camera record of interference fringes. This document describes a Matlab-based program which guides the user through the process of converting these fringe data to a velocity surface. The data reduction is of the "fringe trace" type, wherein the changes in velocity at a given position on the line are calculated based on fringe motion past that point. The analyst must establish the fringe behavior up front, aided by peak-finding routines in the program. However, the later work of using fringe jumps to compensate for phase problems in other analysis techniques is greatly reduced. This program is not a standard GUI construction, and is prescriptive. At various points it saves the progress, allowing later restarts from those points.

Acknowledgments

A variety of people have contributed to the work underlying this white paper. Wayne Trott mentored most of the people at Sandia who have used Line VISAR. Lalit Chhabildas challenged me to persist in making sense of spall void dimensions, and of material strength as well. Tom Ao and Dan Dolan were both available to help me out of tight spots with the Matlab programming. Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Table of Contents

Figures

LineVISAR: A Fringe-Trace Data Analysis Program

1.0 Introduction

1.1 Using this document

The primary goal of this document is to provide a tutorial in using the LineVISAR program. The Introduction primarily provides background information for those less familiar with the instrument. Chapter 2 provides basic facts about the program ("Expectations"), and a capsule set of instructions identical to the helpfile available within the program. It also includes 3 sample data reductions that the reader can work through, illustrating the various capabilities of the program. Finally, Chapter 3 details the specifications, limits and area for later improvement for the program.

1.2 Review - Standard VISAR

The VISAR, or Velocity Interferometer System for Any Reflector [1], was originally developed to convert the Doppler shift of laser light returned from a point into a velocity history. The basic design is shown in Fig. 1.1 and discussed in more detail by Barker [2] and others. Essentially, the VISAR beats Doppler-shifted light against same light slightly delayed by an etalon, producing fringes with beat frequency proportional to sample acceleration. An eighth-wave plate delays one polarization by a quarter phase relative to an orthogonal polarization,

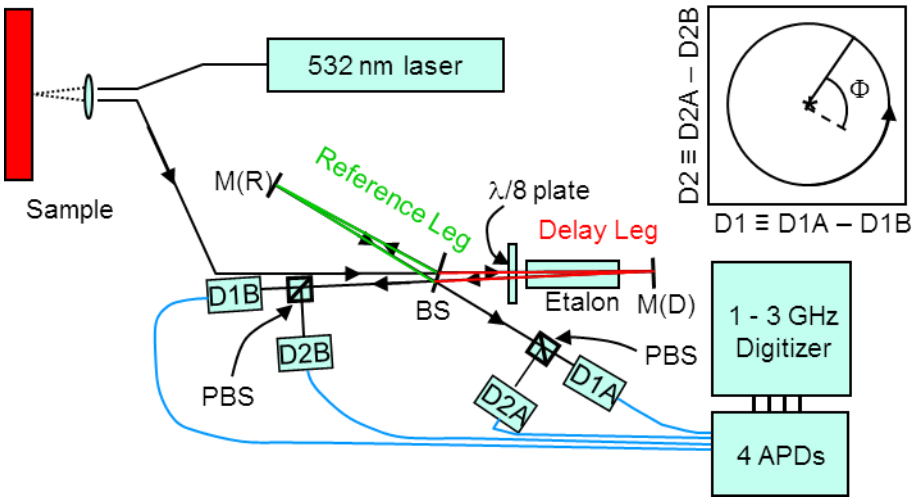


Figure 1.1. Single-point push-pull VISAR. APD is avalanche photodiode; PBS is polarizing beamsplitter, M(D) is mirror (delay leg), M(R) is mirror (reference leg), BS is beamsplitter. In the inset (upper right), the center of the Lissajous circle, corresponding to the zero-light condition, is labeled as an asterisk.

allowing decomposing the output into two phases 90 degrees apart (D1B vs. D2B). The push-pull VISAR developed by Hemsing [3] utilizes light reflected by the beamsplitter as well as transmitted, allowing recording of the other two 90 degree points of the quadrature (D2A and D2B). Differencing D1A and D1B to give D1 ("Data 1"), and similarly with D2, an analyst can plot D1 vs. D2 to obtain a circle plot known as a Lissajous figure, where the angle traversed by the phase point is $\Phi(t) = (4\pi\tau_0/\lambda_0) *$ velocity (where τ_0 is the optical delay in the etalon). The radius of this circle is proportional to the contrast, and under ideal conditions is equal to the intensity of the light returned from the target. The velocity increment required to traverse the Lissajous circle ($\Phi = 2\pi$) is 1 VPF = $\lambda_0 / 2\tau_0$. In recent years, fiber optics have generally been used to carry light to the target and back to the interferometer.

1.3 Review - Line-Imaging VISAR/ORVIS

Consider a variant on the VISAR, where an illuminated line on the sample is imaged through the interferometer onto the input slit of a streak camera (Fig. 1.2). Initially built by Hemsing [4, 5], this allows the deduction of velocity as a function of time and position. Since only one of the quadrature phases is used, the table design is simplified by the omission of the eighth-wave plate and the polarizing beamsplitters. A free-beam laser delivery and return is used instead of fiber optics (although recently some systems have been designed to successfully eliminate speckle problems even though fiber optics are employed for part of the light delivery to the target [6]).

The term ORVIS (Optically Recorded VISar) is often used to convey the fact that a streak camera is used for recording.

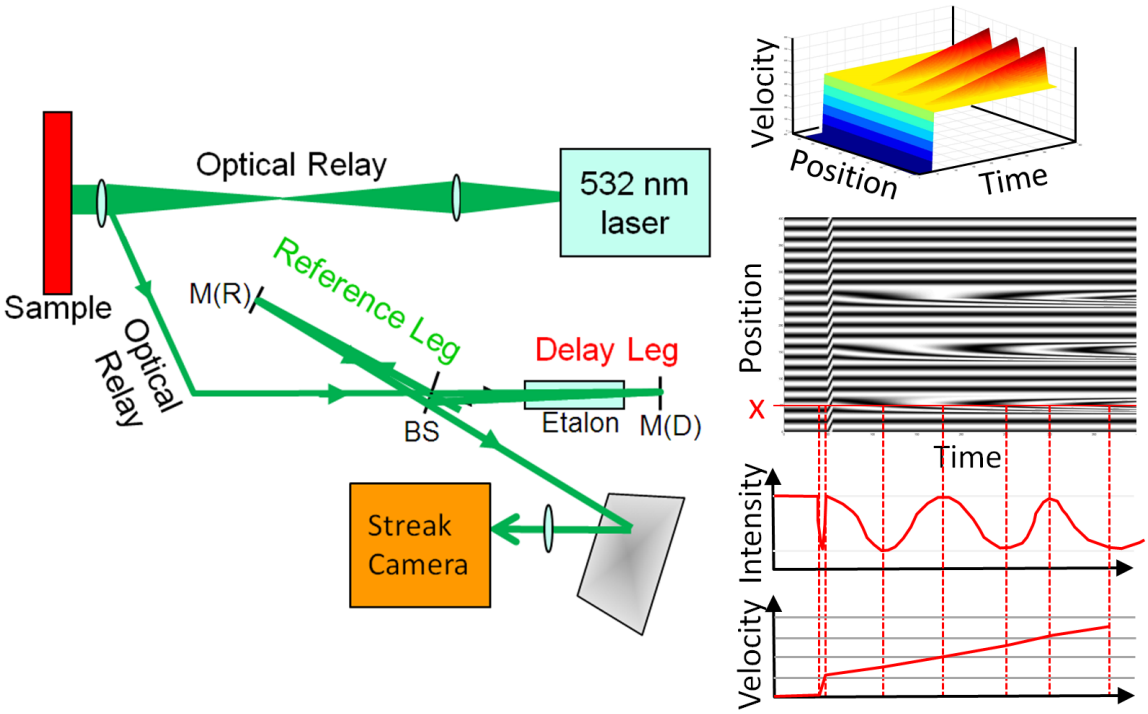


Figure 1.2. (Left) Conventional line-imaging ORVIS. M(D) is mirror (delay leg), M(R) is mirror (reference leg), BS is beamsplitter. (Right) Derivation of velocity profile from streak camera record at position marked by red "x."

The system is normally tuned (beamsplitter rotation) so that ~15 – 30 fringes are visible prior to any surface motion. In this way, the fringes may be seen as contour maps of a surface $A(y,t) \equiv v(y,t) + (By+C)$, where $v(y,t)$ is the velocity, y is the position on the line, t is the time, and B and C are constants determined by the beamsplitter rotation. The contour interval is 1 VPF $= \lambda_0 / 2\tau_0$. B and C (modulo 1 VPF) may be determined from the pre-arrival portion of the streak camera record.

Other variants, such as the Mach-Zehnder interferometer, produce essentially the same streak camera images.

1.4 Data reduction for Line-Imaging VISAR/ORVIS

At Sandia, several programs have been developed to convert the TIF-format output from the streak camera into a velocity surface $v(x,t)$. The first, developed by Trott and O'Hare (VISAR 2K Prep), employs a "poor man's quadrature VISAR" method [7] wherein intensities through time are measured at four constant positions corresponding to ¼ phase increments in the pre-motion streak camera record. The program they prepared is based on National Instruments software, but unfortunately has not been ported to current computer systems. Most of the Sandia work done to date has employed this program for data reduction.

A more recent program by Ao [8], processes the streak camera image by performing a Fourier transform in line position space, windowing the transform function to filter out noise, back-transforming, extracting the phase, subtracting the underlying background phase, and finally correcting by the velocity-per-fringe multiplier to obtain the velocity surface. This program is based on Matlab®, so is better prepared for future porting to other computer systems. It still requires enhancements to make tractable the task of finalizing complicated velocity surfaces (see discussion of fringe jumps below).

The first method intrinsically blurs spatial resolution by pulling in data from a fringe width to deduce the velocity along a constant-position lineout. Typically, this would be ~ 5% of the overall line length. The second method, inasmuch as the Fourier transform is windowed, also does some blurring. It is also more susceptible to being tricked by a large change in velocity across a small increment of position at constant time. Both of these methods require analyst decisions in three areas: (1) spatial and time calibrations (e.g. time per pulse for a comb generator and millimeters per fringe prior to first motion), (2) correction for camera bowing or tilt (generally using a static image), and (3) the introduction of fringe jumps where rapid velocity changes occur. It is this third correction type that requires the most care, time, and understanding of the expected result. Frequently, slight changes in the underlying phase will result in an integer difference in the number of fringes being added by the algorithm at one location versus another. The analyst must untangle this.

Other programs have been developed which extract fringe center positions (e.g., Mastin and Ghiglia [9], Fisk et al [10]), from which it is possible to then deduce velocity as a function of time and position. As well, hand-reductions using this method are possible. There is a distinct trade-off between degree of automation and robustness in handling spatial variations.

The present program requires interactively defining fringe positions (constructive and destructive interference), then uses those to establish phase evolution at all position-time points in the region of interest, which can then be converted to velocity. It linearly interpolates the

phase at each point from the adjacent fringes at the same time coordinate. The primary user effort is in defining the fringe positions; various tools are provided to facilitate this.

1.5 Time resolution limits for Line-Imaging VISAR/ORVIS

For time resolution, we are concerned not only with how well an event can be identified with a pixel number, but also how well the timebase of the full record is established. The timebase is generally established by a time marks from a calibrated comb generator along the side of the streak camera image (roughly 20 marks over the time window of the streak camera), together with an "impulse dot" serving as a time point reference to allow tying the timebase to some "zero time" such as impact on a gas gun.

With care, it is possible to measure the interval between two *sharp* features (each identifiable to +/-2 pixels) to 0.1% of the streak sweep time with a 4K x 4K CCD. As shown in Fig. 1.3, this may be used to measure shock transit times. This process may require correcting for any nonlinearity in the camera timebase. The strength of this is that method is that shock transit times of order 100 ns may be measured to a precision of 20 ps, which is substantially better than to 500 ps available with digitizer-recorded VISAR or the 50 – 100 ps available with PDV.

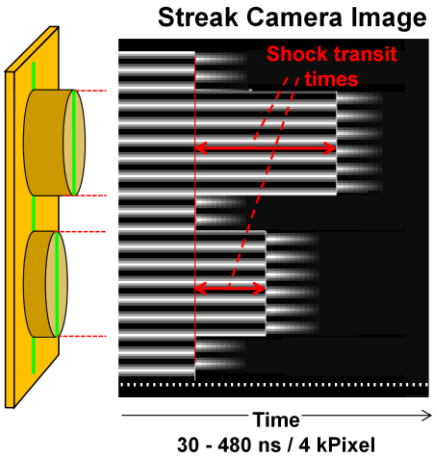


Figure 1.3. Method for measuring shock transit times to high precision with line imaging ORVIS (simulated record).

2.0 Using the Program

2.1 Expectations

Most of this section is a tutorial discussing the process of analyzing three sample data sets. As well, it includes a summary set of instructions (also available when the program is launched).

In general, the user can expect to:

1. Launch the program from within the Matlab console, after setting the working directory to where the data files are found and using the ADDPATH command to tell Matlab where the LineVISAR program is located. Then type "LineVISAR" at the Matlab prompt.
2. Be guided by information prompts throughout.
3. Supply the basic shot parameters (line length, fringe constant, comb generator time spacing and impulse dot time). There are shortcuts available (e.g. supplying the total time window and start time). These can be changed later in the reduction if desired.
4. Choose whether the main objective of this run is to measure time intervals on the streak record, to derive velocity surfaces, or both.
5. If the goal is to derive velocity surface, spend some time interacting with the program to mark the fringes, repair poor fits to the fringes, and number the fringes. The bright fringes (constructive interference) are done first, then the dark fringes (destructive interference).
6. Save intermediate stage reductions, and therefore not have to redo work already done when returning to continue reduction later.
7. (Opt.) Use a preshot fringe image to correct for common image distortions.
8. Supply fringe jumps as needed in the final velocity records (although this process is hopefully less onerous than with other data reduction methods because most of that work is done up-front during fringe definition).

2.2 Summary of program flow

Note: This text is also available as a Help button upon starting program execution.

This program guides you through the line VISAR analysis process. This data reduction is of the "fringe trace" type, wherein the changes in velocity at a given position y on the line are calculated based on fringe motion past that point. The analyst must establish the fringe behavior up front, aided by peak-finding routines in the program. However, the later work of using fringe jumps to compensate for phase problems in other analysis techniques is greatly reduced. This program is not a standard GUI construction, and is prescriptive. At various points it saves the progress, allowing later restarts from those points. It's often convenient (but not necessary) to use a separate directory for each data set.

This program is run from the Matlab console. Set the working directory as the directory with the streak camera image files, type "addpath '<location of LineVISAR program files>' ", then type "LineVISAR".

If beginning a new analysis, the program takes you through the following steps. Common restart points are noted; restarting from a save file avoids repeating steps prior to that point.

(1) Open the shot image file (normally a TIF file). If you have a preshot image file (recommended), open that as well. Choose image rotation and flipping so that time runs left-to-right. At the next step you have the option of redoing this until it's right. For many of the below steps, the user is asked to define positions or polygons on the streak images or other plots. You can enlarge the image to full screen to make this job easier.

(2) Follow the prompts to establish the timebase, spatial scale and velocity-per-fringe. You can enlarge the image display to full screen if desired. Try to keep your clicks within the image; although the program checks for compliance with image boundaries, Matlab routine ginput can behave erratically with clicks outside the image area.

(3) Optional: Select breakaway planes for determining transit times. This feature is intended for use where the target contains a sample on a baseplate, with the illuminated line spanning the sample and part of the baseplate on either side of the sample. Two methods are available: Eyeball estimates of breakaway points for each plane (close-ups of the region of interest are provided to make this process more precise) and threshold determination of when the fringe pattern has changed enough to correspond to an arrival (better when the arrival is sharp). In either case, the transit time and breakaway plane fits (linear and quadratic) are given before you leave the module. There is also the option to save the breakaway planes as Excel files (time versus position, all referenced to pixels). In a later release, it will be possible to use these planes for fringe jump locations.

(4) Establish the bright fringes (constructive interference). If many fringes are similar, you can define those together by defining the top and bottom fringes (see below) and specifying the total number of fringes prescribed this way. The rest of the fringes are established one-by-one. For each fringe, draw an open-ended polygon around it with mouse clicks (generally, the open end is to the left, although it also works to the right). The program then draws the fringe using a running-average maximizing routine for each time. You can redo the fringe if you made a mistake, either by specifying a new polygon or directly drawing the fringe with the mouse. The order of the fringes is unimportant at this stage; you can reorder them later. You can also replot the fringe pattern atop the streak image if the figure is getting cluttered (eliminates the old polygons and numbers the fringes). As well, you can remove unwanted fringes, then add new ones, join two fringes into one (for the case where a fringe wanders off the edge of the image, then back on) and edit portions of fringes. To facilitate editing fringes, the user can zoom in on selected parts of the streak camera image. These actions can be repeated until you have the fringe arrangement you want (at which time you can click "Done With Bright Fringes").

You may want to save the progress as soon as you are satisfied with the fringes before you click "Done With Bright Fringes"; this is a good restart point for later.

5) Order the fringes to number from top to bottom. The automatic numbering (option "Number from Top to Bottom") often will do the job. Occasionally fringes starting at much later times may need special handling ("User-Supplied Numbering"), for which you will need to type the order of the fringes reflecting the top-to-bottom sequence (e.g. 4 1 2 3 5). Click on "Use Present Order" when you're satisfied. The destructive interference ("dark") fringes are then constructed between the bright fringes, using a running-average minimization calculation at

each timepoint. You have the option of repairing any of these dark fringes, and can zoom in on portions of the fringe image to facilitate this process. The work you've done is now saved as a second restart file.

(6) If you supplied a preshot file in the beginning, follow the prompts to define the fringes on that image. Only the vertical position of your clicks really matters. The fringe slopes are used to deduce camera rotation and tapering (pincushioning and barreling may be added later). The program takes a few moments to process the file (a "Processing" window appears briefly), then provides a dialog box for saving the corrected image, and finally draws the corrected image with the fringes also corrected to match.

(7) Select the Region Of Interest (ROI). Fringes that don't run the full length of the image are extrapolated to the left or right margins (note that this can create inaccuracies in the velocity later, especially if they cross other fringes).

(8) If desired, follow the prompts to provide impact tilt information (linear or quadratic) to be subtracted from the timebase.

(9) Click on two separate timepoints in the portion of the record before motion begins (only the time coordinates matter here). The time interval between these will be used for computing a pre-motion correspondence between phase and position. The program now computes the velocity surface, plots it in pixel coordinates, and saves the program status.

This is another common restart point.

(10) You can now manipulate the velocity surface and save the plots to files. Options include adding/removing fringe jumps (specifying jump location by mouse or by typing in pixel locations, together with width of jump (pixels) and velocity jump as number of fringes), selecting a rectangle for measuring spatially averaged velocity histories, saving the progress, saving the velocity histories, and examining the plots on the screen, saving plot images, etc. (these plots are locked unless you elect to examine them because the program is not written as a callback-driven GUI). Plots can be displayed in units of pixels or mm (microns) and μs (ns).

(11) When desired, quit the program.

This program detects the user's monitor size and uses this information to decide how to position the plots. If you start Matlab, then reduce the monitor resolution (e.g. to connect to a projector), this feature is defeated and you may not be able to see the entirety of your plots. It is better to change the resolution before launching Matlab..

2.3 Example 1 – transit time measurement

This example uses the image files "TransitTimeExample_Shot.tif" and "TransitTimeExample_Preshot.tif." Start the program and indicate that you are starting with a "New" analysis. Open the shot and preshot image files (there are prompts in the upper left of the dialog window). Accept the defaults for image rotation and flipping. This image should look like the left panel of the following:

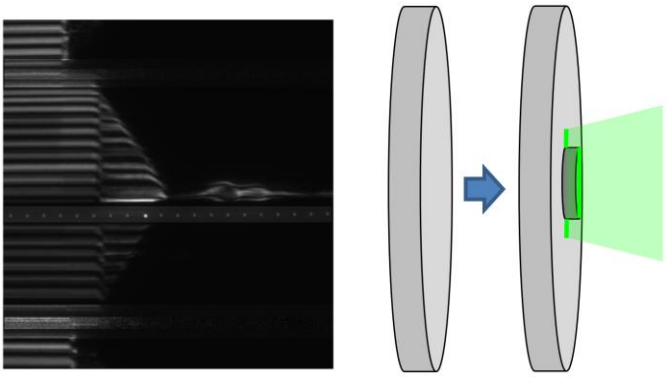


Figure 2.1. (left) Transit time example streak image; (right) shot geometry

The shot setup includes a tantalum sample (6.06 mm diameter and 0.709 mm thick; [100] orientation) mounted atop a diamond-turned aluminum disk, impacted by an aluminum disk at 1.03 km/s. The timing comb (100 ns spacing) and impulse dot have been placed in the middle of this image (rather than at one edge) to maximize the area available for fringe records on the line portions returned from the substrate. The VPF is 0.4838 km/s.

The impulse dot lies in the same line as the comb timemarks, so the automated finding of comb timemarks can't be used. Click in the first and last timemark and specify 1900 ns (there are 19 100 ns spacings). The automated peak-finding of the impulse dots works well. Note that the program marks the location of mouse clicks with labels on the image.

Select the option to measure shock transit time. For these low-pressure shots (where the waveforms are not sharp arrivals), it is best to select the option "Picking breakaway points by eyeball." The other option ("Intensity variation thresholds") can be better when there is a sharp arrival resulting in a drastic and immediate intensity change.

For the first breakaway, you will name it (for example, "Al/vac" for an aluminum/void interface). You are able to specify one or more working regions for each breakaway (necessary because the baseplate breakaway has portions below and above the sample). Each working region is a rectangle, and is specified by clicking on two opposite corners. Once you have specified a working region, the program zooms in to that region. You can now click on a family of points specifying the breakaway (left clicks of the mouse). Press "Enter" when done; at that time the program asks you to verify the points you have supplied. If these are not OK, redo them. If they are, you now return to a view of the entire plot, with the points and working region just defined labeled.

After defining both interfaces, your plot appears as shown in Fig. 2.2, together with a summary of the breakout fit parameters and the deduced transit time.

If you use the "Intensity variation thresholds" option, the program will display the family of points determined for the fit, and allow you to adjust the threshold and redo the fit.

At this point, you can proceed into a velocity surface calculation or quit.

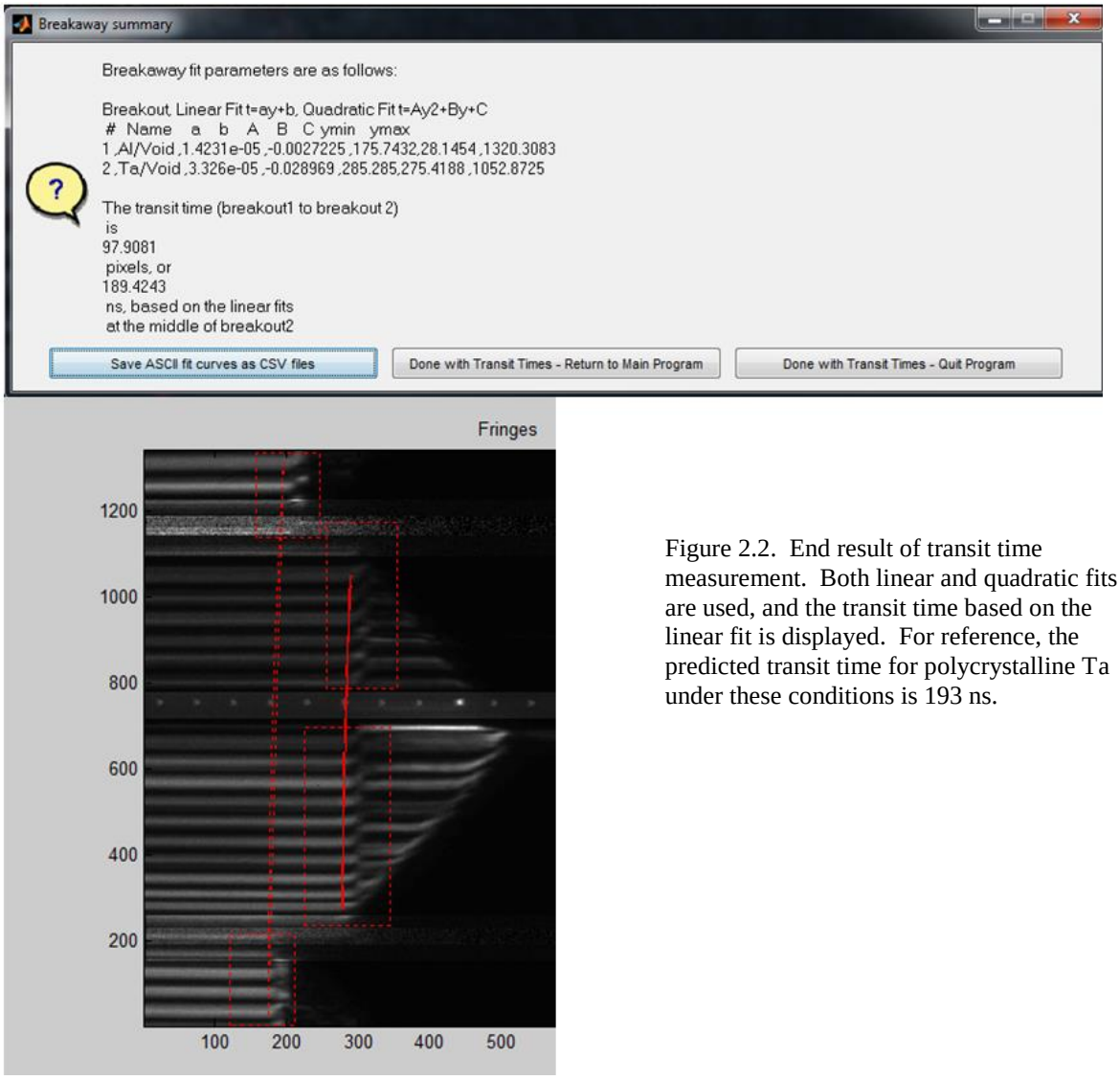


Figure 2.2. End result of transit time measurement. Both linear and quadratic fits are used, and the transit time based on the linear fit is displayed. For reference, the predicted transit time for polycrystalline Ta under these conditions is 193 ns.

2.4 Example 2 – repetitive fringe patterns

This example uses the image files "251 Shot.tif" and "251 Preshot.tif." Start the program and indicate that you are starting with a "New" analysis. Open the shot and preshot image files (there are prompts in the upper left of the dialog window). Accept the defaults for image rotation and flipping. For the timemarks, select "Find timemarks automatically." Then specify a bounding rectangle that includes all ten timemarks but does not include the impulse dot (it is a very narrow rectangle). The program will insert a yellow "+" sign atop each timemark. Each timemark represents 500 ns. The impulse dot may be found by specifying a small rectangle around it as well, and the program will insert a cyan "+" atop that marker. At this point the display will appear as shown in Fig. 2.3.

The next goal is to define the fringes. For the present case, there is very little spatial variability in the fringes, so we select the option to define several fringes at once. The instructions in the dialog boxes are self-contained. However, it is useful to develop the following practice in defining a polygon enclosing a fringe:

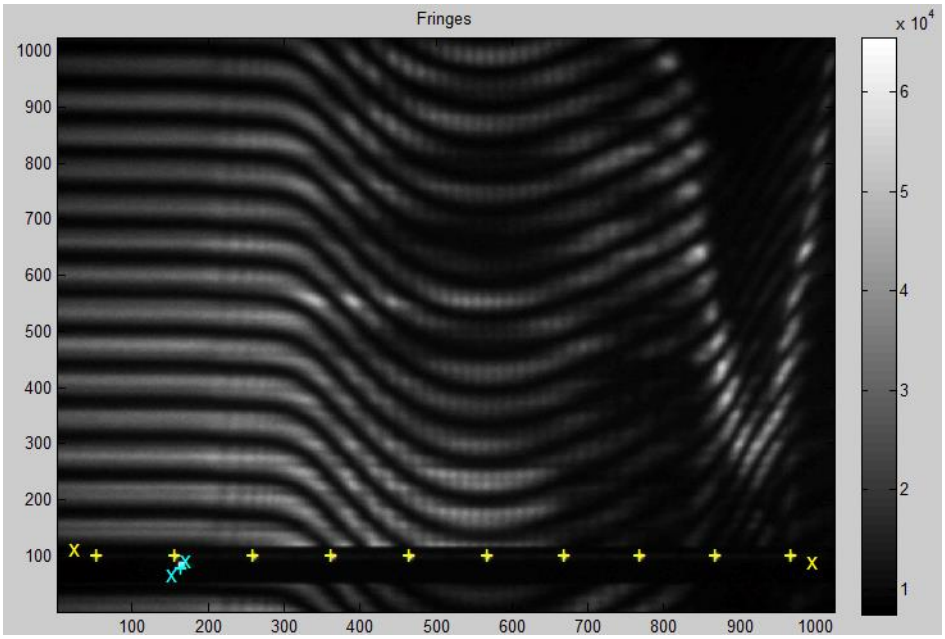


Figure 2.3. Shot 251 streak image with timemarks and impulse dot selected.

1. Click on a point just above the left-hand end of the fringe, still on the streak image.
2. Click on a series of points to define a curve just above the fringe.
3. At the right-hand end of the fringe (which may or may not be at the right side of the streak image), click on a couple of points close together, then do the same just below the right hand end of the fringe.
4. Now work your way back to the left hand end. With each click, the program updates the polygon with another red line segment.
5. Do not close the polygon at the left end.

Typical "polgonology" is illustrated in Fig. 2.4, which shows the two main methods for defining polygons bounding the space in which the program searches for a bright fringe (fringe from Example 3 image). After you have finished defining the top and bottom fringes, the program takes a minute to fit the intermediate fringes. The end result is shown in Fig. 2.5.

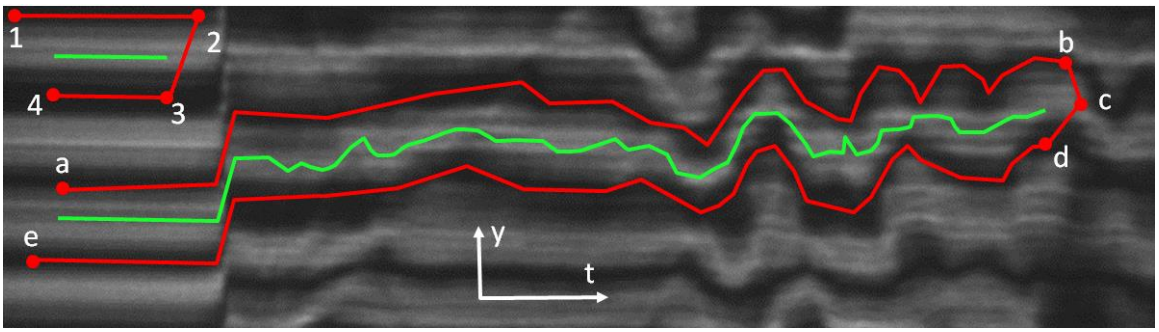


Figure 2.4. Construction of polygons for bounding fringe regions. The most common type of polygon has 5 or more points. Points *a* and *e* represent the first and last points input. Point *c* has the largest time value. The (green) fringe constructed begins at time $t = max(a, e)$ and ends at time $t = min(b, d)$. The case of a quadrilateral (upper left) is treated specially, as illustrated.

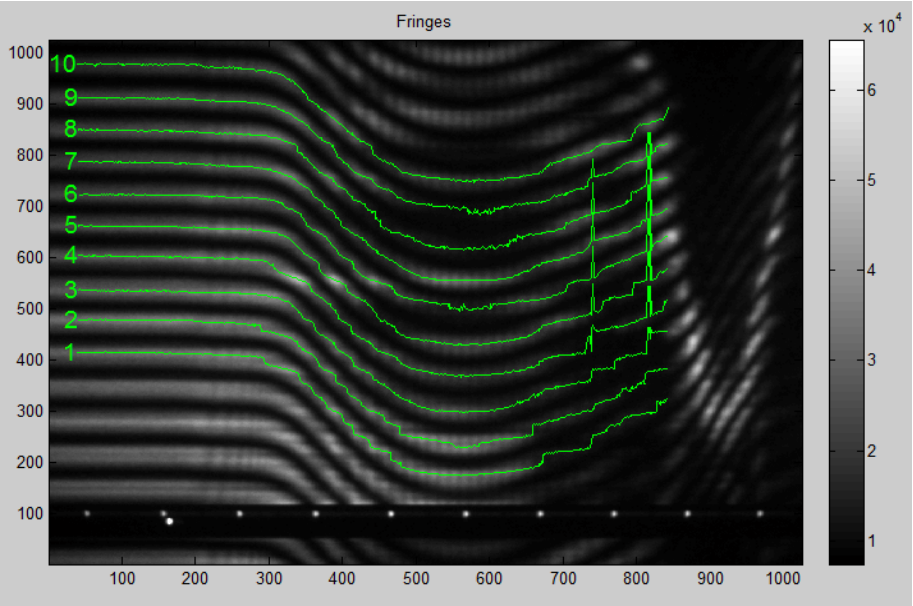


Figure 2.5. Shot 251 streak image with first group of fringes selected.

You can now proceed with a combination of adding new fringes and repairing fringes. An example of a repair process is shown in Fig. 2.6. Repairs can be done with or without zooming in, and can involve extending the fringe to earlier and/or later time or replacing a piece of the fringe as displayed (in green) with a repair (shown in red until the repair is accepted). Repairs are done strictly by mouse clicks and do not rely on automated center-finding.

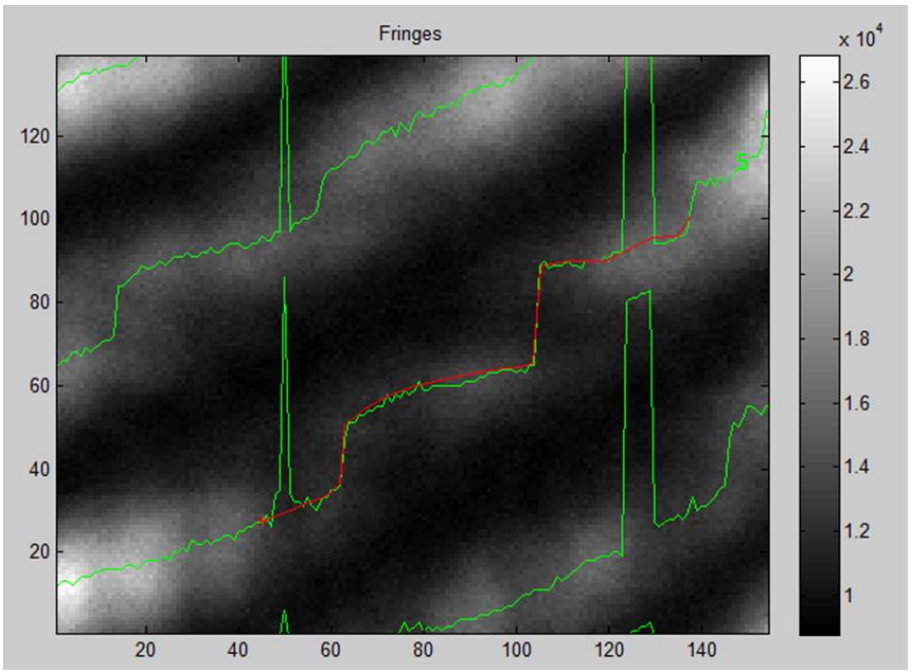


Figure 2.6. Process of repairing Fringe 5, using "zoom in" option. Note that fringe 5 is labeled near right edge of this portion of the image.

After a mix of repairing and adding fringes, the pattern shown in Fig. 2.7 is obtained. For illustration, fringes have been extended across the time mark strip.

In defining bright fringes, the program assumes you will supply a polygon, and only afterward offers the option of defining the fringe by mouse clicks. Sometimes it is necessary to just define a polygon somewhere (e.g. a quadrilateral as in Fig. 2.4), then decline the resulting fringe and *then* input the fringe you really want by mouse clicks.

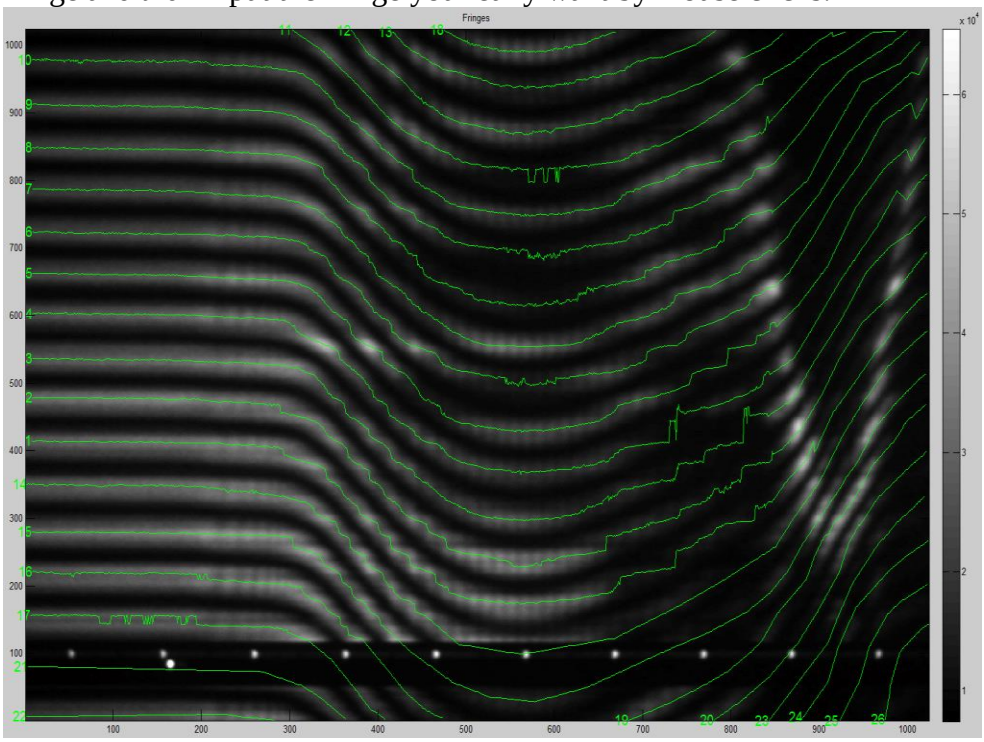


Figure 2.7. Completed bright fringe patterns, prior to joining and renumbering.

At this point, it is necessary to tie together fringes which are separate occurrences of a single fringe. In Fig. 2.7, several fringes drop off of the bottom of the image, then reappear with different numbers farther to the right. For example, fringes 16 and 19 need to be joined. Once two fringes are joined, the resulting combination is assigned a single number and the other fringes are renumbered. This process is repeated until all required joinings are completed.

Next, the fringes must be renumbered to increase upward (number 1 is at the bottom). Several options are available for doing this. The automatic sequencing works in some cases (where all fringes reach the left side of the image). In other cases, you must use the manual sequencing. Here, type in the number of the bottom fringe first, then work your way up to the topmost fringe (space or comma delimiters). The plot with renumbered fringes confirms the result. If you made a mistake, repeating the command with the correct number list should correct it.

Now you have the option of saving your work (strongly recommended).

The next task is to construct and verify the dark fringes (destructive interference). The program automatically calculates these fringes, constrained to lie between adjacent bright fringes, and numbers each one paired with the bright fringe immediately below it. In well-defined regions of the streak image, it is likely no repairs will be needed. If the eventual region of interest is completely well-behaved, the user can move on. If not, it will be necessary to repair dark fringes. This is done similarly to the earlier repairs on bright fringes. The completed result is shown in Fig. 2.8.

If there is substantial work to be done here, it is advisable to "save and quit," then re-enter the program. At that point, restart and read in the file you just saved.

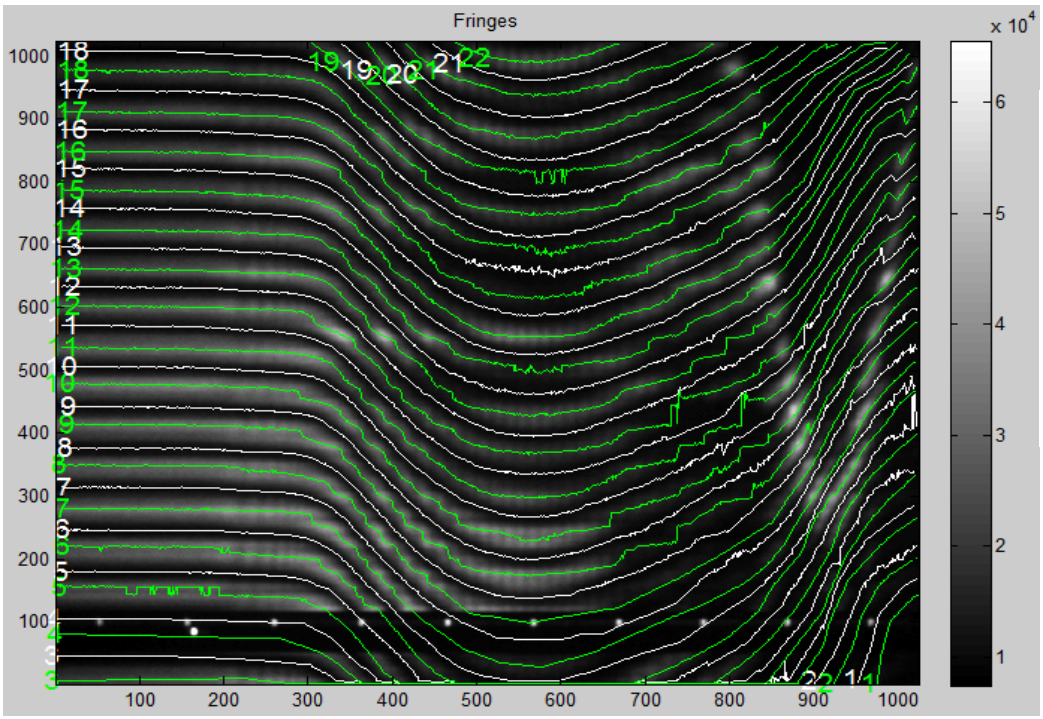


Figure 2.8. Completed dark fringe patterns (white curves) plotted together with bright fringes (green curves).

When you have finished editing the dark fringes, save the file, restart the program as before and choose the option that you are done editing the dark fringes. The program asks for a save file name, and it is wise to choose a different one than before because restarting from this file takes you directly into the next step (the default filename complies with this).

Now it is time for image corrections (if you have supplied a preshot file). A small plot of the preshot images is presented (Figure 2.9). Click above and below the top fringe (echoed as yellow "x"s), then repeat for the bottom fringe. The program constructs a fit to each of these (shown by green lines), then asks how many fringes are there to fit. For the example shown here, 14 is the response. If you confirm that these fits appear satisfactory, the program corrects the image for camera rotation and wedging. After saving the corrected image file and a save-work file, it is time to proceed to velocity calculation.

After re-confirming the shot parameters (VPF, timing, and line length), you have the option of correcting for impact tilt. The prompts are self-explanatory. The next step is selecting a time interval prior to first motion via two mouse clicks (only the time coordinates of these locations matter). This is used to construct a zero-motion phase map.

At this point the program calculates the velocity surface. It prompts you for the name of a save file for this velocity surface (a Matlab .mat) file), then offers options for manipulating the velocity surface. Note: The velocity plot may be hidden behind another plot (find it from the taskbar). You must choose "examine" to rotate or zoom on the surface. Other options include adding fringe jumps and selecting a rectangular region for constructing an average velocity plot (velocity averaged over the spatial coordinate). Due to the Matlab constraint of 3 options per pushbutton box, some of these options require proceeding through 2- 3 layers of pushbuttons (self-explanatory).

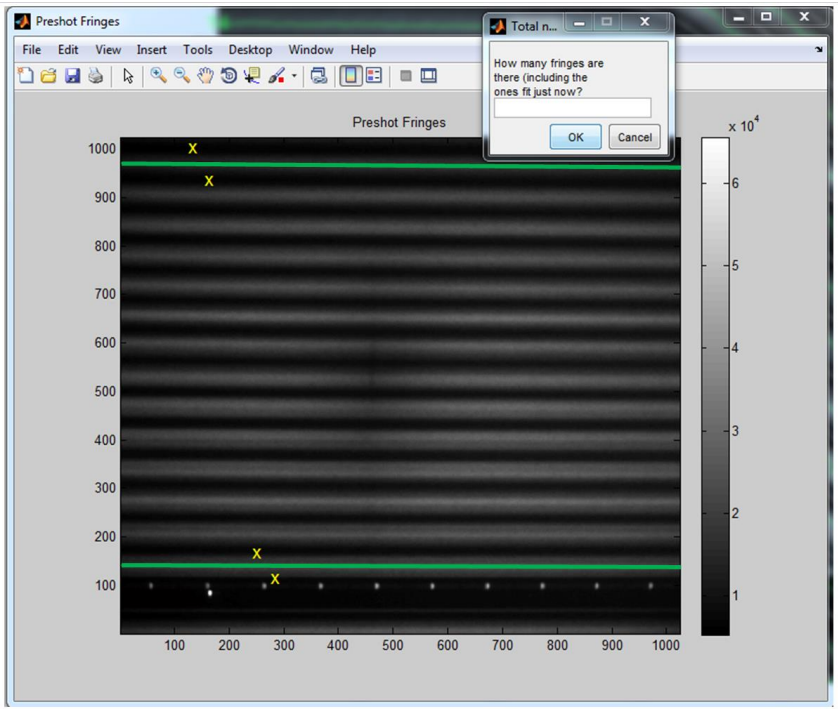


Figure 2.9. Preshot image with top and bottom fringes fit. Yellow "x" markers indicate upper and lower boundaries input by the user.

The waveform produced for the present problem is shown in Fig. 2.10.

An averaged velocity profile is shown in Fig. 2.11 (averaged over the red rectangle region on the fringe record). Note that regions for averaging are defined on the fringe plot instead of the velocity plot. The same holds true for fringe jump locations (please do not quit this plot while working). All such work is done in pixel coordinates, with plots converted to real coordinates in time and position only as needed. This averaged curve can be saved as an Excel file (including time, mean, ±1-sigma and extreme value columns).

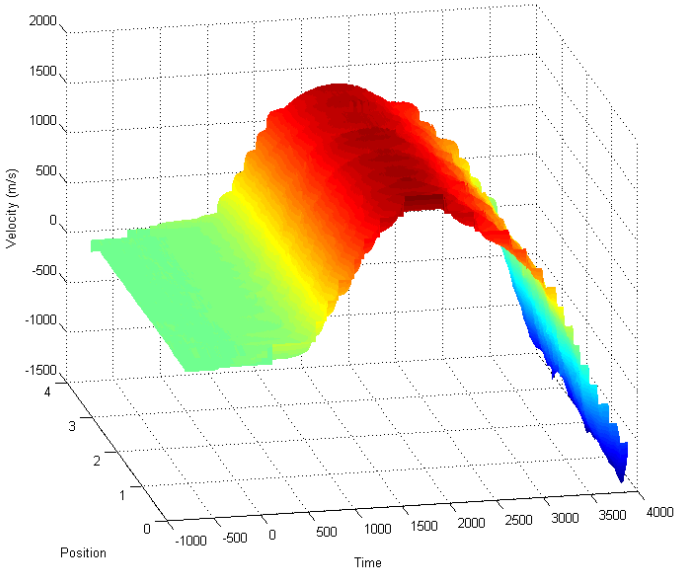


Figure 2.10. Velocity surface. Time and position coordinates used can be pixels or real units (e.g., ns and mm).

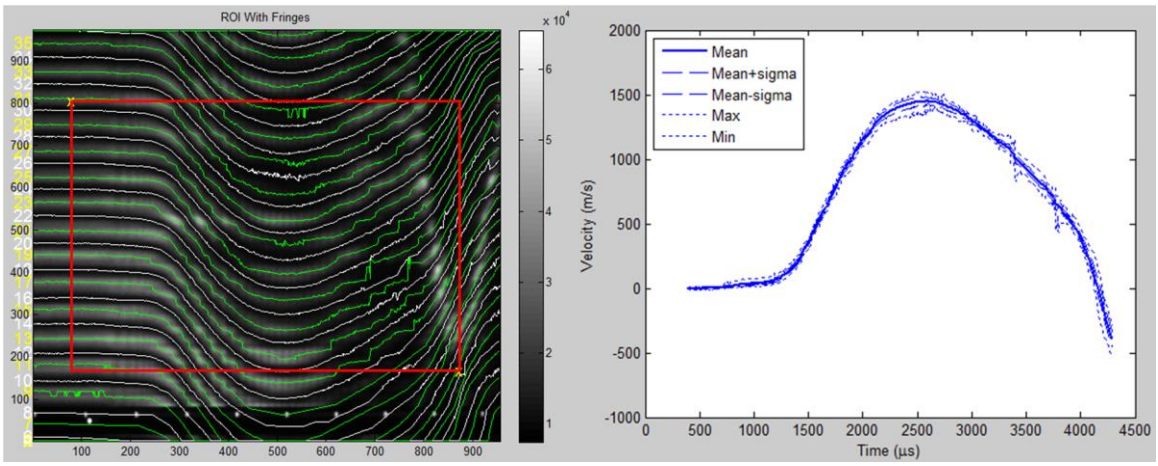


Figure 2.11. Velocity averaging tool. The region is selected on the fringe plot (left) by mouse clicks on opposite corners and shown in red. The plot on the right shows spatially averaged velocity as a function of time. The plot on the left uses pixel coordinates; that on the right can use pixel or real coordinates (real coordinates chosen for present example).

The "save" command here saves not only a Matlab .mat file, but also .xlsx (Excel 2010) versions of the velocity surface, the time axis and the position axis for export to other plotting and/or analysis programs.

2.5 Example 3 – More complicated fringe patterns

This example uses the image files "198 Shot.tif" and "198 Preshot.tif." This is for a copper bicrystal. The VPF is (-)139.45 m/s, the comb dot spacing is 500 ns and the line length is approximately 16 mm.

Start the program and indicate that you are starting with a "New" analysis. Open the shot and preshot image files (there are prompts in the upper left of the dialog window). Accept the defaults for image rotation and flipping. Proceed to define the fringes as in the previous example, except that here they must be added one-by-one. Save and quit the program as soon as the bright fringes are added, then restart it from the restart file. Reorder the fringes to number from bottom to top. Proceed to define the dark fringes. In this case, the dark fringes do not need any repairs because (1) the region of interest really doesn't include any uniformly dark areas that will cause dark fringe instability. This image should look like Fig. 2.12.

Several cautions apply:

First, when we later select a region of interest (ROI), any fringes which do not reach the left of right hand edges of this will be extrapolated at constant position to those edges. This can result in fringe crossing. That is why bright fringes number 1 and 2 are "repaired" to reach the left-hand (t =0) edge, even though they are drawn across regions with no fringe information. If an ROI is selected which doesn't include fringes 1 and 2 at all, this can be disregarded. It is safer to use an ROI which doesn't get too close to the top and bottom of the streak camera record.

Second, it is best to ensure that fringes do not cross during shock loading. The algorithm appears tolerant of this, but may misbehave on occasion.

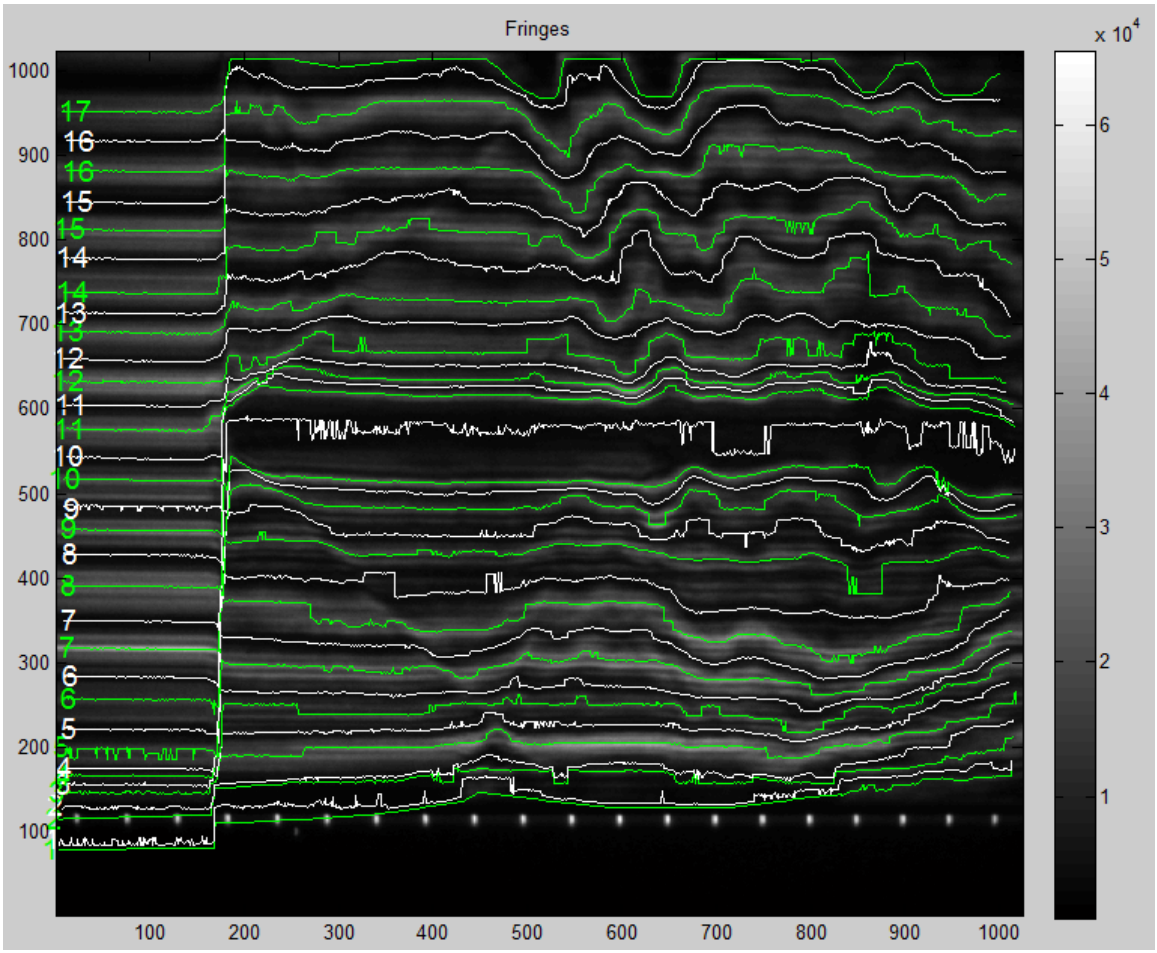
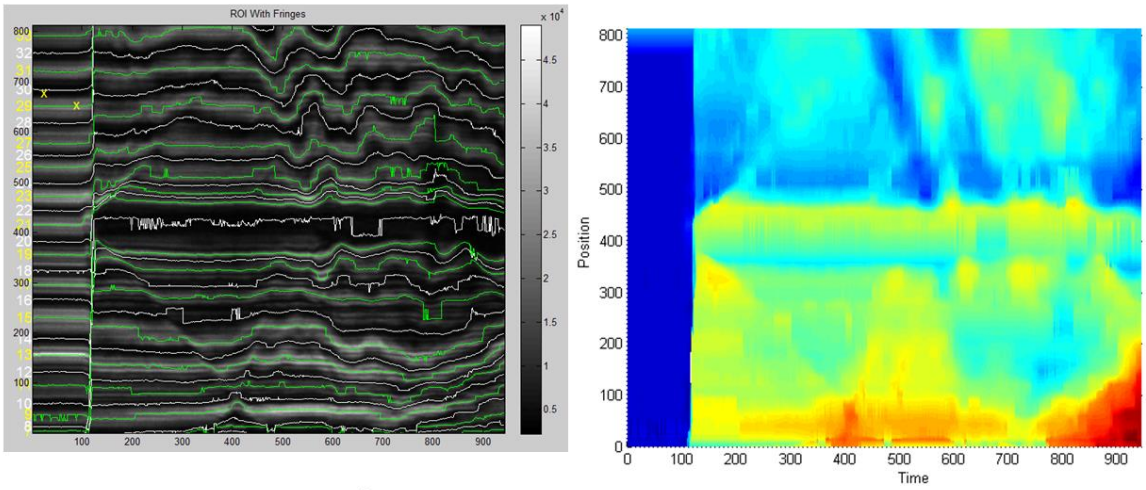


Figure 2.12. Shot 198 streak record and fringes.

Again, the analysis proceeds as in the previous example. After the image corrections are applied, the ROI is selected, and any corrections for impact tilt are applied (none in this case), the velocity surface is computed and plotted. Occasionally that plot is not presented on top of the other Matlab windows, so it may need to be pulled up with help from the taskbar. Figure 2.13 shows this result.

A fringe jump appears indicated for the top (early time) of the image. In this case, it is of magnitude (-1)·VPF and width 1 pixel. The position is chosen with a mouse by clicking on the ROI fringe plot (see left side of 2.14) or by typing in the pixel coordinates of the endpoints. After effecting the jump, the plot appears as in Fig. 2.14. Up to 100 jumps can be used at a time. These jumps can be selectively edited or deleted later, and the velocity surface can be saved with jumps included.



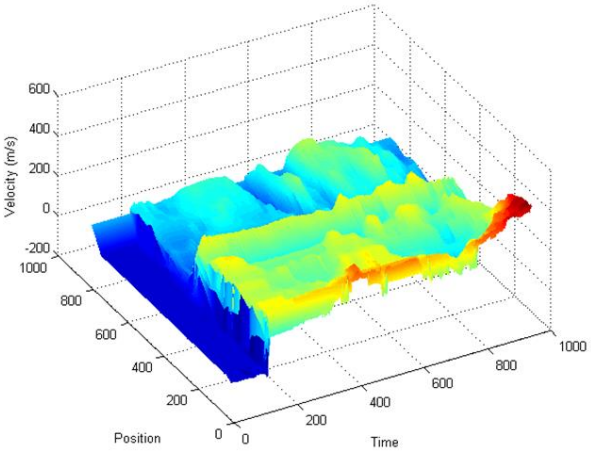


Figure 2.13. Region of interest plots of fringes and velocity (pixel coordinates used here).

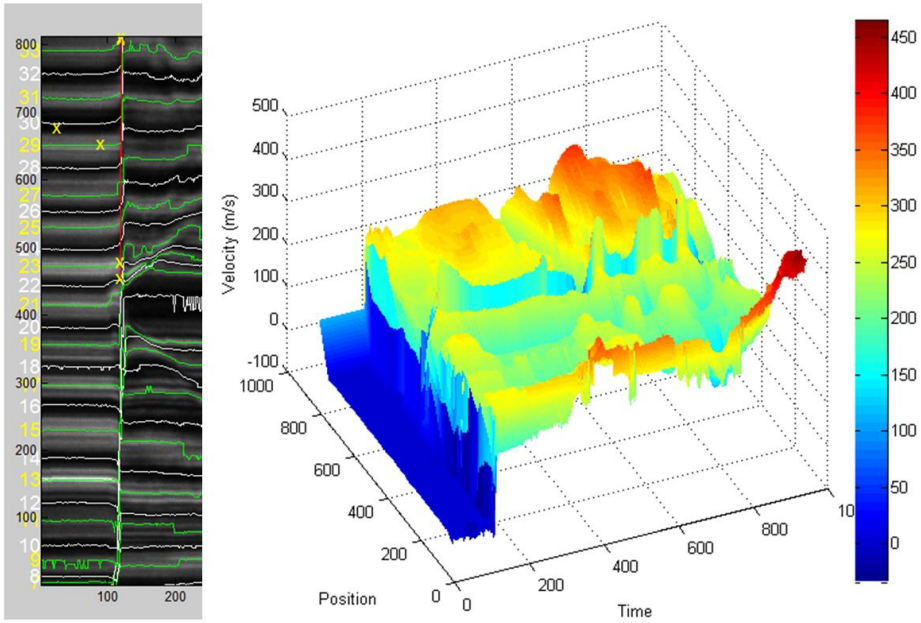


Figure 2.14. Velocity surface with fringe jump added over interval shown by red line on left inset.

3.0 Specifications, limitations, and future work

3.1 Specifications

Operating system: Operates under Matlab. It was written under R2013b on Windows 7, but should be portable to other systems.

Plot sizes and monitor resolution: The streak camera records generally default to fullscreen (the exception is the preshot image). For user convenience, fringe repairs may be done on zoomed-in portions of the streak record. Plots of the velocity surface start of as partial screen, but during inspection may be enlarged to full screen if desired. Reducing the screen resolution during program execution (e.g. hooking up a projector) can cause problems.

Number of fringes: Limited to 120 at the present time (bright plus dark); can be adjusted easily in routine ParamSetup.

Maximum number of points in polygon bounding fringe:

Number of velocity jumps: Limited to 100 at present (adjustable); each is defined by a line segment, number of fringes, and ramp width in pixels.

Streak camera image size: Not specifically limited; also, does not have to be square.

Image format: Tagged image files (TIF) presumed. Brightness and contrast should be adjusted prior to using this program.

Outputs: Matlab .mat save files, Excel .xlsx spreadsheet files of velocity surface, timebase and position axis; also of averaged velocities. For plots, the use of screen capture or Snip is easy, although after exiting the program or during manipulation of the velocity surface, the Matlab plot save utility may also be used.

Maximum number of interfaces for transit time measurements: 10

Maximum number of regions for each interface for transit time measurements: 8

3.2. Porting results to other programs

There are three principal ways outputs from this program can be used elsewhere:

(1) Plots may be manipulated during program operation while the "inspect" option is in effect, including resizing, saving via the Matlab saveplot utilities on the top bar. The same is true after program termination with any on-screen plots. As well, data can be extracted from the plots as follows:

1. Select the arrow in MATLAB figure toolbar.
2. Select the graphical object of interest (in this case, the line).
3. Use the "get" command to access the object's properties.

```
x=get(gco,'XData');
y=get(gco,'YData');
```

4. For image objects, the height is stored as color data.

```
z=get(gco,'CData');
```

Of course, screen capture tools (e.g. Snip) can also be used to save plot images.

(2) The velocity, time and position arrays are saved as Excel files. They may be re-imported into Matlab or used in Excel. Excel 2010 can handle these velocity output files; earlier versions may not be able to due to line length limitations.

(3) The restart files can be read directly by Matlab outside of the LineVISAR program via the command:

```
fringe=load(restartfilename);
```

The structure elements of interest are as follows:

`fringe.y` (the spatial coordinate, relating pixel number to position)
`fringe.time` (the time coordinate, relating pixel number to time)
`fringe.velocity` (the velocity array, indexed as (time,position))
`fringe.roi` (the cropped image array, indexed as (position,time))

3.3 Known shortcomings

Fringes which appear midplot in the region of interest (ROI): Fringes which appear at the edge can be treated by approximately drawing them at the edge and keeping the inaccurate parts outside of the ROI. Fringes which appear in the middle of the line (e.g. the sample in Fig. 1.2) cannot be handled at this time. A future release will be able to reduce such a data set.

Fringes which disappear midplot: Similar to fringes which appear midplot. This also includes "island" fringes and "loop" fringes", which correspond to strong positional variation in velocity.

Fringe repair tools: While the user can extend a fringe, (s)he cannot shorten it. Workaround: Delete and re-define the fringe.

Plot manipulation: While responding to prompts (Matlab routines questdlg), the user cannot resize plots or use any of the Matlab plot commands (zoom, save, object selection & retrieval, inserting labels, etc.). That is why the "inspect" option is offered during analysis / viewing of the velocity surface plots (note that the user can manipulate *any* plots currently defined while analysis / viewing is available).

Serial question dialogs and lack of help at various points: This is limited by the number of buttons available in a *questdlg* diaplog box. Another option is now available, and may allow better layout of these dialog boxes in a future release.

Defining blocks of similar fringes: This task can only be done at the beginning of the fringe definition process. This may be corrected in a future release.

3.4 Future work

Future revisions to this code will center on two areas: (1) allowing the processing of fringe patterns including loop fringes and other structures that include fringes beginning and ending in mid-image, and (2) better user interfaces.

References

1. Barker, L.M. and R.E. Hollenbach, *LASER INTERFEROMETER FOR MEASURING HIGH VELOCITIES OF ANY REFLECTING SURFACE*. Journal of Applied Physics, 1972. **43**(11): p. 4669-4675.
2. Barker, L.M., *The development of the VISAR, and its use in shock compression science*, in *Shock Compression of Condensed Matter-1999, Pts 1 and 2*, M.D. Furnish, L.C. Chhabildas, and R.S. Hixson, Editors. 2000, Amer Inst Physics: Melville. p. 11-17.
3. Hemsing, W.F., *VELOCITY SENSING INTERFEROMETER (VISAR) MODIFICATION*. Review of Scientific Instruments, 1979. **50**(1): p. 73-78.
4. Hemsing, W.F., et al., *VISAR - LINE-IMAGING INTERFEROMETER*. Ultrahigh- and High-Speed Photograph, Videography, Photonics, and Velocimetry 90: Eighth in a Series, ed. L.L. Shaw, P.A. Jaanimagi, and B.T. Neyer. Vol. 1346. 1990, Bellingham: Spie - Int Soc Optical Engineering. 133-140.
5. Hemsing, W.F., et al., *VISAR: line-imaging interferometer*. Shock Compression of Condensed Matter - 1991. Proceedings of the American Physical Society Topical Conference, 1992: p. 767-770770.
6. Malone, R.M., et al., *Overview of the line-imaging VISAR diagnostic at the National Ignition Facility (NIF) - art. no. 634220*, in *International Optical Design Conference 2006, Pts 1 and 2*, G.G. Gregory, J.M. Howard, and R.J. Koshel, Editors. 2006, Spie-Int Soc Optical Engineering: Bellingham. p. 34220-34220.
7. Trott, W.M., et al., *Measurements of spatially resolved velocity variations in shock compressed heterogeneous materials using a line-imaging velocity interferometer.* AIP Conference Proceedings, 2000(505): p. 993-998.
8. Ao, T., *LIVA: A data reduction program for line-imaging ORVIS measurements* 2009, Sandia National Laboratories Report, SAND2009-3236.
9. Mastin, G.A. and D.C. Ghiglia, *DIGITAL EXTRACTION OF INTERFERENCE FRINGE CONTOURS.* Applied Optics, 1985. **24**(12): p. 1727-1728.
10. Fisk, G.A., G.A. Mastin, and S.A. Sheffield, *DIGITAL IMAGE-PROCESSING OF VELOCITY-INTERFEROMETER DATA OBTAINED FROM LASER-DRIVEN SHOCK EXPERIMENTS.* Journal of Applied Physics, 1986. **60**(7): p. 2266-2271.

Distribution

Internal:

1	MS 1186	Dept. 1640	D. G. Flicker
1	MS 1189	Dept. 1641	T. Mattsson
1	MS 1189	Dept. 1641	R. W. Lemke
1	MS 1189	Dept. 1641	H. L. Hanshaw
1	MS 1189	Dept. 1641	K. Cochrane
1	MS 1195	Dept. 1646	J. F. Benage
1	MS 1195	Dept. 1646	C. S. Alexander
1	MS 1106	Dept. 1646	T. Ao
1	MS 1193	Dept. 1656-1	S. Payne
1	MS 1193	Dept. 1656-1	R. Hacking
1	MS 1195	Dept. 1646	J. Brown
1	MS 1195	Dept. 1646	J-P. Davis
1	MS 1195	Dept. 1646	D. H. Dolan
1	MS 1195	Dept. 1646	M. D. Furnish (10)
1	MS 1195	Dept. 1646	M. D. Knudson
1	MS 1195	Dept. 1646	S. Root
1	MS 1195	Dept. 1646	C. T. Seagle
1	MS 1195	Dept. 1646	J. L. Wise
1	MS 1195	Dept. 1647	G. T. Leifeste
1	MS 1195	Dept. 1647	R. J. Hickman
1	MS 1195	Dept. 1647	W. D. Reinhart
1	MS 1195	Dept. 1647	T. F. Thornhill
1	MS 1454	Dept. 2554	M. A. Cooper
1	MS 1454	Dept. 2554	B. A. Jilek
1	MS 0899	Technical Library (electronic copy)	

Sandia National Laboratories