

Interactive processing: Geometry manipulation

Robert G. Clapp

ABSTRACT

Header manipulation for regularization, registration and quality control is often a time-consuming task with 3-D datasets. These manipulation tasks are often performed multiple times before achieving the desired result, each requiring a large amount of data to be read in with very few operations performed on each byte read. The problem can be made more tractable by reading in a random subset of the headers. Operations such as rotating, windowing, and gridding can then be performed interactively. With each processing step a record is created. These records can then be used to process the entire dataset, requiring only a single read and write of the volume.

INTRODUCTION

SEP has a long history of writing interactive viewers for regularly sampled multi-dimensional volumes (Ottolini, 1982; Biondi and van Trier, 1993; Clapp et al., 2008; Clapp, 2010). It has also dabbled in interactive processing using a range of platforms from Sunview (Claerbout, 1991) to Xtpanel (Cole and Nichols, 1992, 1993) to AVS (Clapp and Biondi, 1994). These attempts have met with limited success because they often did not provide a significant advantage over batch processing in terms of efficiency or ease of use.

The last 15 years have seen a significant change in the relative speeds of disk, memory, and compute power, with the former lagging far behind. In addition the size of seismic data volumes has grown by at least an order of magnitude. This brings into question the paradigm of running a batch program, QC'ing with a viewer a small percentage of the volume, then repeating with new parameters or proceeding to the next step. A different paradigm is to put the viewer as the central player, minimizing the amount of disk IO in exchange for redoing computations. This paradigm can be particularly effective in manipulating headers, which generally involve a low number of operations for each byte read accessed.

In this paper I introduce a new tool, `qthead`, for interactive header manipulation. The application allows the user to perform all of the basic header key functions in SEP3D (Biondi et al., 1996). The user works on a random subset of the headers, rotating, windowing, and gridding until a satisfactory result is achieved. The application stores all of the commands. These commands can then be applied to the entire volume, greatly reducing the required I/O.

DESIGN

In this section I will begin by describing the SEP3D tools that `qthead` replaces. I will then move into a brief description of the various classes in `qthead` and their functionality.

SEP3D tools

There are several SEP3D header manipulation applications in SEPIb that `qthead` attempts to replace. `Headermath` allows the creation of keys based on mathematical expressions using constants and other key values. This includes the ability to rotate the coordinate system by entering a rotation matrix based on `x` and `y` values. `Window_key` subsamples the dataset based on ranges of values for different keys. `Sort3d` allows the user to overlay a regular grid on irregular key locations. Finally, the programs `Stack3d` and `Infill3d` create a regular dataset from an irregular volume using the grid created in `Sort3d`.

FUNCTIONALITY

There are three windows in `qthead`. The main window gives a graphical display of one or more headers. The desired header can be selected from a pull down menu at the top of the display (see Figure 1). In this display the user can window a range of headers by drawing a box using the left mouse button. The right mouse button can be used to draw a line which will be used to rotate the data. After rotation the selected line will be horizontal in the display.

The second window (see Figure 2) displays all of the actions that have been performed on the data such as rotating and windowing. The user can undo an action by selecting the action above it in the list using the left mouse button. The application will then rerun all of the processing steps above and including the selected action. The bottom portion of the window is the gridding parameters that the user has selected for the dataset. These can be set in the final window.

The first two rows of the final window (see Figure 3) allow the user to specify exact windowing parameters for the data and an exact rotation angle (instead of using the mouse approach used in the main window). In addition, the user can create new headers keys by entering the new key name and its formula using constants and other key values. The final row allows the user to add a gridding axis to the dataset. The minimum and maximum of each key are provided by the program, the user adds the sampling then uses the `go` button to create the axis. Once an axis has been created the user has the option of choosing to change the main window display from a cross plot to a multi-dimensional histogram (see Figure 4).

The main window panel provides additional functionality using drop-down menus. The user can store all of the commands in the history buffer to a file. A history buffer

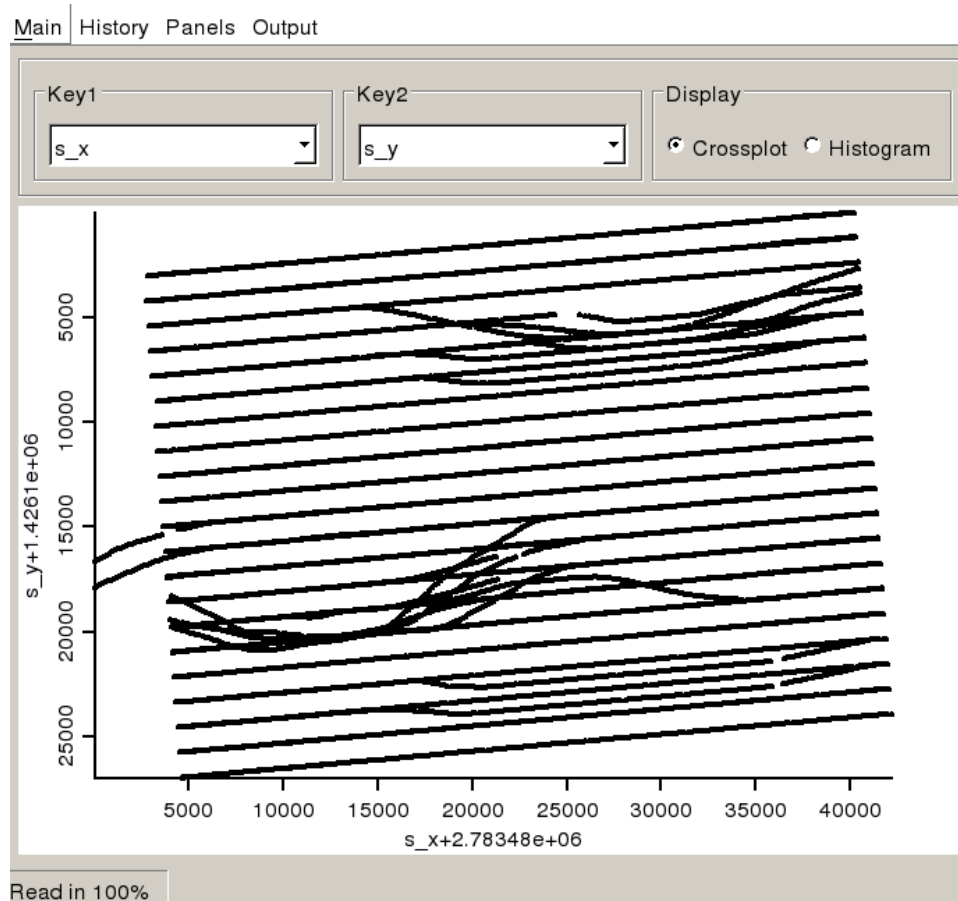


Figure 1: An example of the cross-plot view. In this case we see the source locations displayed. Note the ability to select the headers to be displayed at the top of the figure along with the ability to switch between a cross-plot and histogram view.

Figure 2: A view of the history panel. Each command can be clicked on, erasing all future processing steps. The display is then recreated by running all commands before and including the selected command.

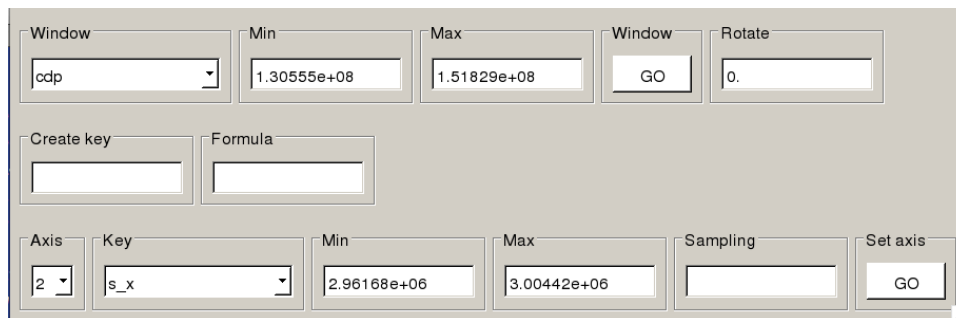
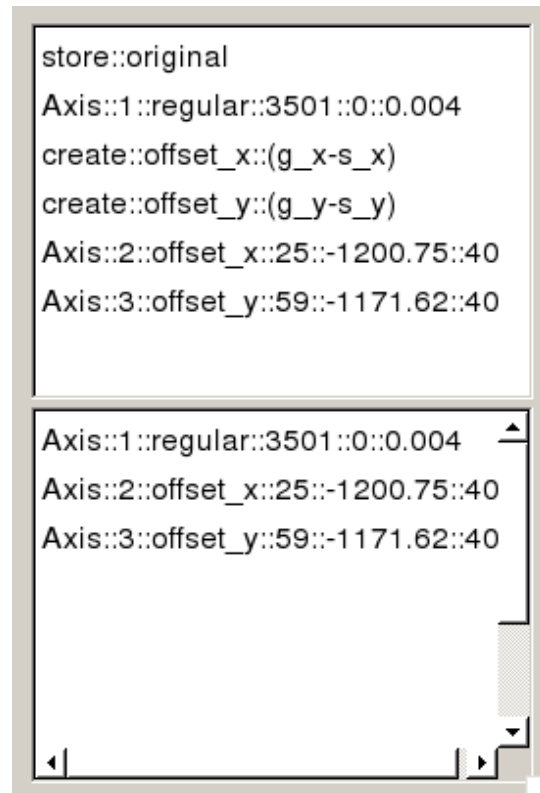


Figure 3: The header manipulation panel. From this panel keys can be windowed and rotated using the keyboard inputs. In addition new keys can be created by entering a formula based on constants and other keys. The bottom row allows gridded axes to be created from a selected key.

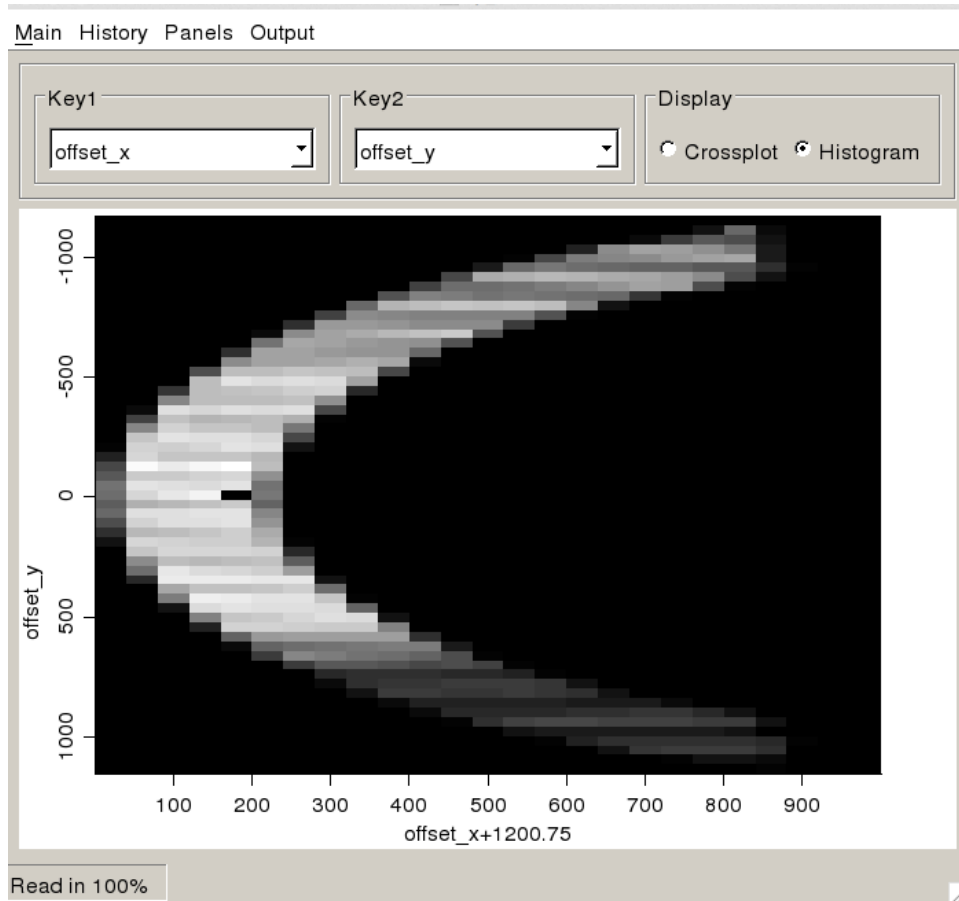


Figure 4: An example of the histogram view. In this case two offset keys have been created and the data binned at 40m. Darker colors indicate more traces falling into a bin.

can also be read from disk, and the actions in it performed automatically. The user also has several different output options. The user can output all of the SEP3D commands that will produce a SEP3D dataset equivalent to the current display. The user can also directly create the volume using the `Create regular volume` or `Create irregular volume`. The difference between these two options is whether to create a regular cube (requires that gridding axes have been created) or an irregular volume.

Classes

The application, `qthead` is written in C++ using the QT¹ toolkit. The most basic building block for the application is the `headers` class. The `headers` is the basic storage mechanism for the headers. It has the ability to return an array of floats, doubles, or integers for all the headers currently stored in memory. The `io` class is the interface to read and write header values to disk. Currently only the ability to read/write SEP3D volumes is implemented but expansion to support SEG-Y, SU, etc. would require minimal effort. The `my_data` replicates most of the functionality of `Headermath` and `Window_key` such as creating keys, rotating, and windowing. It is the basic interface to the headers for all of the other components.

The fact that the entire header volume can not be assumed to fit into main memory resulted into several design decisions. Reading in a random subset of the data to interactively manipulate is an effective visualization strategy but is not sufficient to create an output volume to use for further processing. The need to rerun all of the commands on the entire dataset led to the creation of the `action` generic class. This class includes virtual functions (that result in calls to the `my_data` class) to perform desired actions.

The `history` class operates as the control unit for `qthead`. It keeps track of all the actions that user has performed. It has the ability to run a series of actions on a given `my_data` object which is useful both for when the user wants to remove a processing step and when the final output volume needs to be created. This ability to run a series of processing steps on an arbitrary header volume allows the `history` class to replace the functionality of `Sort3d`, `Stack3d`, and `Infill3d`. It can loop through a given dataset, reading manageable subsets and performing a series of actions before writing out the final dataset.

The most sophisticated class is the `io_thread` class. This class, a separate thread created when the application is started, is responsible for reading in random subsets of the headers. It starts by reading random subsets of the headers in chunks of several MB. After each chunk is read it applies all of the current actions using the functionality of the `history` class. Once all of the allotted header memory is used it stays dormant until one of two conditions is met. If the user requests that an action is undone. This results in the current data volume is abandoned, random subsections are reread, and the remaining processing steps run on it. The second reason for the

¹<http://www.qt-proj.org>

thread to be awoken is when a windowing operation is performed. After a windowing operation the thread will read in more of the headers, applying all of the current actions, until either the entire volume has been read or the maximum allotted space is again reached. Disk I/O can be a significant bottleneck, particularly given the random read approach used by `qthead`. To get around this problem we use Ecoram (Clapp, 2009), a type of solid state memory with a read rate of around 2GB/s, which seems sufficient for this application.

FUTURE WORK

A similar interactive approach can be used for many of the pre- and post-imaging steps now done at SEP using batch processes. Instead of reading in and processing an entire volume, only the portions of the dataset needed for the current view need reside in memory. This is a subtle but important difference. Processes such as NMO operate on a single trace, but doing a velocity scan requires all of the traces at a given CMP location. As a result each routine must have an awareness of what range of data is required to display a given subsection. As the number of processes increases the application must be able to keep track of a potentially expanding tree of data required to read into main memory to produce a given view.


CONCLUSIONS

I present an interactive 3-D geometry manipulation application, `qthead`. The application allows the user to rotate, window, and grid 3-D data interactively. The user builds a processing flow that is performed on a randomly subsampled portion of the data stored in memory. This processing flow can then be performed on the entire dataset to produce a final output volume or additional random subsets as more memory becomes available. This approach cuts down substantially on 3-D geometry processing costs and is applicable to other low-op pre- and post-imaging steps.

REFERENCES

- Biondi, B., R. Clapp, and S. Crawley, 1996, Seplib90: Seplib for 3-D prestack data: SEP-Report, **92**, 343–364.
- Biondi, B. and J. van Trier, 1993, Visualization of multi-dimensional seismic datasets with CM-AVS: SEP-Report, **79**, 1–12.
- Claerbout, J. F., 1991, Interactive one dimensional seismology program ed1D: SEP-Report, **71**, 293–294.
- Clapp, R. G., 2009, Visualization and data reordering using ecoram: SEP-Report, **138**, 297–304.
- , 2010, Hypercube viewer update: SEP-Report, **140**, 229–232.
- Clapp, R. G. and B. Biondi, 1994, Iterative velocity model building for 3-D depth migration by integrating GOCAD and AVS: SEP-Report, **80**, 635–643.

- Clapp, R. G., D. M. Chen, and S. Luo, 2008, Hypercube viewer: SEP-Report, **134**, 179–192.
- Cole, S. and D. Nichols, 1992, Xtpanel: An interactive panel builder: SEP-Report, **75**, 497–520.
- , 1993, Xtpanel update: Interactivity from within existing batch programs: SEP-Report, **77**, 409–414.
- Ottolini, R., 1982, Interactive movie machine user’s documentation: SEP-Report, **32**, 183–195.



**I DON'T ALWAYS CALCULATE GREEN'S FUNCTIONS.
BUT WHEN I DO, I MAKE SURE TO USE THE ONE-WAY WAVE
EQUATION IN FREQUENCY-WAVENUMBER DOMAIN.**