SeaSeis: A simple open-source seismic data processing system

Author: Bjorn Olofsson, Seabird Exploration, bjorn.olofsson@sbexp.com

Introduction

SeaSeis is a seismic data processing system that I developed over the past six years, and which has been distributed as open-source software for about three years. I believe the system is easy to use and easy to add new modules to, which are the two main objectives I had in mind when seriously starting to write it. "Ease of use" is obviously subjective, but ease to write and bind new modules into should be possible to judge more objectively.
The main appeal of SeaSeis may be for researchers/developers who want to use it as a platform to quickly implement and test newly developed algorithms, but it may also appeal to quite a different type of users: Since SeaSeis provides broad functionality facilitating production processing, such as master flow generation and logging, a small processing company may find it useful to use it to supplement any other available (licensed) systems. In my dreams (and who says these never come true?), SeaSeis will at some point be combined with the strengths and functionality of other open-source processing systems, to form one fully capable system.

Base functionality

SeaSeis is a sequential trace flow system designed for pre-processing of seismic data.
It consists of:

- Batch (base) system that fully manages the trace flow
- Processing flow parser (ASCII text file, example shown in figure 1)
- Command line submission tool (figure 2)
- Around 80 processing modules, each including a short online documentation
- Interactive 2D seismic viewer SeaView (figure 3)
- Functionality for master flow generation, variable substitution, job submission and log generation
- Prototype job flow builder (figure 1)

Current **strengths**/advantages of SeaSeis compared to other systems:

- Stable base system, thoroughly tested on large commercial projects (marine onboard QC)
- Easy to learn for both users and module programmers (knowledgeable in C++), easy to install
- Allows automation of repetitive tasks, by providing good base functionality to run large number of flows for large number of data sets
- Provides logging and other functionality facilitating QC and reproducibility of processing flows[1]
- Platform independent, only requires standard ANSI C/C++ compiler and Java (JRE)
- 2D seismic viewer SeaView has – for a free software – high usability with acceptable performance. SeaView is non-restrictive, i.e. it doesn't define fixed rules about the type and trace headers of the input data
- Modern source code architecture: Object oriented C++ and Java. Writing a program in a modern language does not guarantee good design or maintainability, but it greatly helps in my opinion

Current **weaknesses**/disadvantages of SeaSeis compared to other systems:

- Only provides very limited geophysical capabilities for data processing
- Not straightforward for new users to figure out which modules have been implemented thoroughly, and which ones are mere place holders with rudimentary functionality only
- Not optimised for performance or very large data sets (but any size data set can be processed)
- No parallel processing capabilities

[1] SeaSeis follows the mantra of Seplib and Madagascar (i.e. reproducibility) without the need of – in my personal view infamous – tools such as UNIX "make"

- Graphical user interface for building & submitting jobs only in prototype stage
- Hardly adopted by anyone: Tiny existing user or developer base
- Usability decreases rapidly for modules with complex user parameter dependence

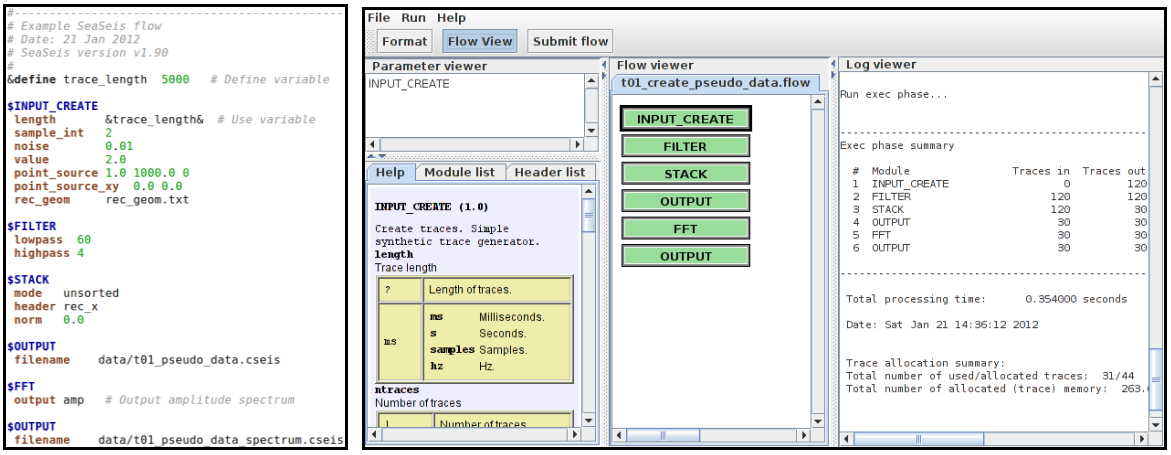

Figure 1: *Example ASCII flow file (left), and prototype job builder (right).*

```
seaseis -h
 SeaSeis job flow submission tool.
 Usage:  seaseis -f <jobflow> [-o <joblog> | -d <joblog_dir>] [-h] [-m <name>] [-v] [-c] [-std] [-p {speed|memory} ] [-g <const_file>] [-s <spreadsheet>]
 -f <flow1> <flow2> ... : File name(s) of job flow(s) to run
 -o [<log>|stdout]      : File name of job log (defaulted to flowname.log if not specified)
                        : Use 'stdout' to redirect all log file output to standard output
 -d <log_dir>           : Name of directory where job log shall be saved
 -s <spreadsheet file>  : Name of file containing spread sheet for building of master flow
 -g <global const. file>: Name of file containing global constants (&define statements)
 -D <name1>[=,:]<val1> <name2>[=,:]<val2> ... : Define user constant <name>, value <val>. No spaces allowed around equal sign.
                                NOTE: The first user constant will be used to create the log file name.
 -ff <flow_out>         : Name of file where job flow shall be saved (only works when -D option is specified).
 -m <name>              : Print help for module <name>.
                        : If no module is found, print help of all modules starting with <name>.
                        : If <name> = empty: Print full help for all available modules.
                        : If <name> = .: Print short help for all available modules.
 -run_master            : Immediately run all flows created from master flow(s)
                        : If not specified, the program will exit after the master flows have been created.
 -h                     : Print this page
 -html                  : Print full help for all modules & standard trace headers in HTML.
 -v                     : Print out version info
 -std                   : Dump all standard trace headers
 -c                     : Check for link problems and consistency of all modules' params(=help) methods.
 -p [speed | memory]    : Set memory policy: Optimised for speed or memory.
 -no_run                : Do not run flow. This option is useful if an individual flow file is generated using option -ff
 -init_only             : Run init phase only.
 -no_verbose            : Do not output information messages.
 -debug                 : Output extensive DEBUG information for trace flow preparation and execution.
```

Figure 2: *Command line arguments.*

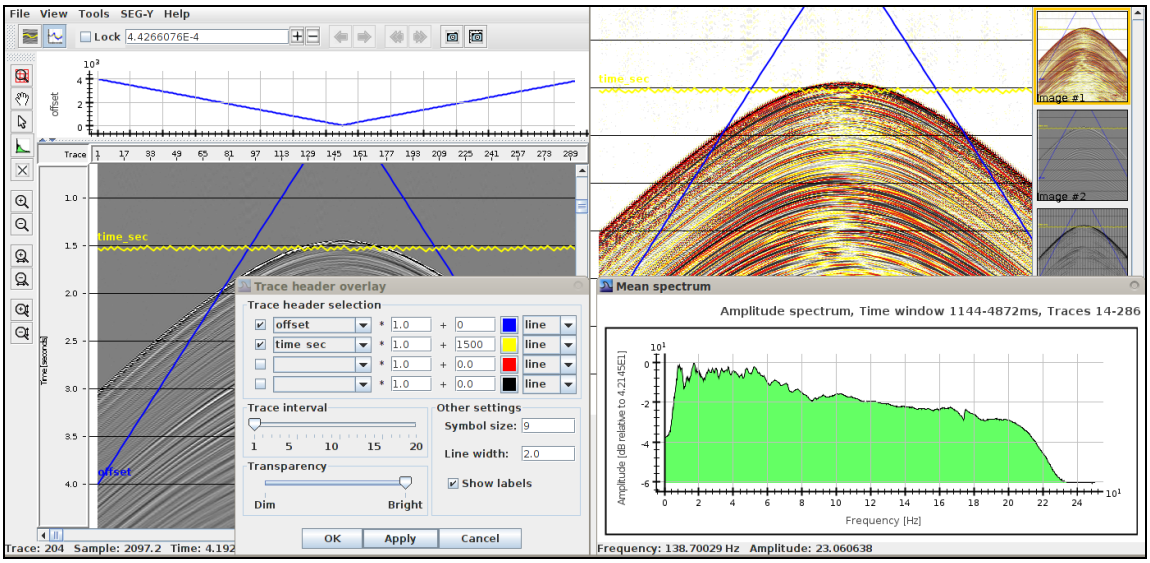

Figure 3: *SeaView 2D seismic viewer.*

Most weaknesses listed above are by choice of priority which has always been ease of use and stability. The underlying design of the SeaSeis system should allow addressing the above issues with minimal time and effort if wanted.

System design

The SeaSeis base system is fully written in C++. Other auxiliary methods are 95% written C++, and 5% in C. Modules are also written in C++ or C, with the exception of a 2D ray tracer which is written in FORTRAN 77. The base system fully manages the trace flow: It parses the processing flow, checks input parameters, passes traces between processing modules, collects ensembles, allocates memory and provides other libraries to a module programmer. The difference to a system like Madagascar is that in the latter case, the base system only provides helper functions without actually managing the trace flow which is done by the UNIX pipe.
Traces currently consist of a trace header and trace data (=the data samples). The trace header expands or shrinks dynamically depending on the number of trace header fields defined by the user (arbitrary number of trace header fields, arbitrary name); the same applies to the length of the trace data (currently restricted to 2^31 number of samples if this can be handled by the computers memory). However, for each individual module instance in a flow, all traces have the same length, with a fixed sample interval. In other words, each module in a flow encounters traces with fixed trace header and trace data length, but can change these arbitrarily before passing them on to the next module. Since traces and other fields in the system are C++ objects, it is easy to extend the "trace definition" and for example pass vectors or other objects through the same interface, without breaking any existing code or modules.

Module design

Each module implements three methods with a standard interface:

Param method	Defines all user parameters, including online documentation.
Init phase	Called once for initialisation, no traces passed through.
Exec phase	Called once for each input trace or ensemble, and one more time for cleanup.

Each of these C++ subroutines is wrapped in a C subroutine in order to facilitate dynamic linking to the executable process in real time when running a job. No compilation takes place during run time.
All objects that are passed through the module method interfaces (such as the seismic data traces) are true C++ objects which get allocated only once by the base system. *Single-trace* modules receive a single input trace and pass on a single output trace (or none). *Multi-trace* modules receive a whole trace gather as input which is defined by the user and module programmer; all traces for the gather are collected automatically by the base system and passed on to the following modules afterwards. The interface is written in a modular way with some possible extensions in mind, such as multiple input/output streams for each processing module. Also, since traces and other fields are C++ objects, it is easy to extend the "trace definition" and for example pass vectors or other objects through the same interface, without breaking any existing code or modules.

Modules

SeaSeis currently has around 80 processing modules. Most of them deal with the logistics of the trace flow and generic manipulation of trace headers. Some other examples follow:
IF, ELSE, ELSEIF, ENDIF allow flow branching.
SPLIT and *ENDSPLIT* allow a creation of a separate branch.
SELECT, KILL, KILL_ENS and others allow omitting traces from flow.
REPEAT duplicates input traces or whole ensembles, for example for parameter testing purposes.
HDR_MATH allows free-form mathematical expressions to manipulate trace headers. Arbitrary trace header fields can be defined and header names can have arbitrary length.
FFT and *FFT_2D* perform forward and inverse fft using the fftw library (included in source distribution, not as an external dependency)

CCP, *ORIENT*, *ORIENT_CONVERT*, *ROTATE*, *HODOGRAM*, *SPLITTING* are specific multi-component processing tools
SUMODULE is a generic wrapper for Seismic Unix modules (externally installed binaries), but it is still experimental.
Several input modules exist to read the following data formats: SEG-Y, SEG-D (most, but not full SEG-D rev 0-2.0 definition), Miniseed, ASCII, and SeaSeis internal format.

User interface

A SeaSeis processing flow consists of an ASCII text file (figure 1). The scripting "language" in which flows are written follows a handful of syntactic rules. User parameter names are only mildly abbreviated, if at all, and hence mostly self-descriptive. This means that flows can be read and parameter options often understood even by a user who has never used SeaSeis before. Users can define arbitrary variables directly in a flow, or supply them via a command line argument or an external ASCII file; these can then be referenced throughout the flow.
Flows are submitted via the batch command line tool (figure 2). A prototype graphical job builder is included in the current version but this needs more programming effort to complete.

2D seismic viewer

SeaView is a basic 2D seismic viewer written in Java (see figure 3); it consists of:

- Generic 2D seismic display graphics engine (CSeisLib) based on the generic Java 2D graphics engine
- Access to SeaSeis code through Java Native Interface (JNI), mainly for the data readers
- SeaView application: 2D seismic viewer
- PlotImage application: High-resolution image generation through SeaSeis module $IMAGE

The CSeisLib class providing the generic 2D seismic display inherits from the standard Java class JPanel and can be used in any graphical application with its full capabilities illustrated by the application SeaView. In other words, it is only a matter of minutes, maximum hours for a Java programmer to produce a graphical application skeleton that can display seismic data, after having had only a superficial glance at the source code. SeaView itself is a demonstrator (not very well written) of the underlying generic capabilities of the library CSeisLib (better written).

Outlook

At the time of writing, there is no specific forward development plan for SeaSeis. In the recent past, I have made opportunistic updates and published a new version about every 6 months, but I will probably not be able to keep up with this pace. Ideally someone else with more time to spare would take over the code base and develop it further, or even better: Integrate it with other open-source processing tools. At least it would be nice if some of its concepts or tools would survive (because I thought hard about them and believe they contain some valuable pieces of knowhow).

Please visit **www.seaseis.com** for more information.