Software framework for geophysical data processing, visualization and code development

by

Glenn Chubak

A Thesis Submitted to the College of
Graduate Studies and Research
in Partial Fulfillment of the
Requirements for the degree of
MASTER OF SCIENCE
in
Geophysics
in the Department of Geological Sciences
University of Saskatchewan

University of Saskatchewan

February 2009

TABLE OF CONTENTS

Acknowledgements

I would like to thank my supervisor Igor Morozov for his commitment to teaching, his enthusiasm as ideas spawned new projects, and perhaps most of all his patience while I finished this thesis. I'd also like to thank Shannon Blyth for all of her hard work and assistance with the programming, Capt. P. Lascelle of Dundurn Canadian Forces Ammunition Depot for his support of the SMS monitoring station and other projects, Brain Reilkoff for putting up with the strain I caused on the IT department and Don Gendzwill for his numerous discussions and suggestions. My committee members have given freely of their time, advice, and expertise – thanks you all. My wife of course is in need of endless gratitude for the support and motivation (most of it good natured) she provided and of course for caring for our children while I was so distracted on other things. I never would have finished without her. To Kaylee and Samuel my incredible children – you are both a wonderful blessing and a source of continual joy.

Table of Figures

List of Acronyms and Glossary

1D, 2D, 3D	One-, two, and three-dimensional, respectively
A/D	Analogue-to-Digital signal converter
ASCII	Standard for text encoding
AVO	Amplitude Variation with Offset
DISCO	Commercial seismic processing system by Cogniseis (now included in Echos by Paradigm)
Focus	Former name of seismic processing system Echos by Paradigm
FreeUSP	Free Unix Seismic Processing system by Amoco
GIS	Geographical Information System
GMT	Generic Mapping Tools (software for mapping and graphics)
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	HyperText Mark-up Language
I/O	Input/Output
IGeoS	Integrated Geoscience Software (formerly called SIA; this name is used in Chapters 1-6)
IRIS	Incorporated Research Institutions for Seismology (U.S. university consortium)
MPI	Message Passing Interface (software parallelization library)
OpenGL	Popular Open-source graphics library
OS	Operating system
PAR4CH	4-channel seismic recording hardware by Symmetric Research
PC	"Personal Computer"
ProMAX	Commercial seismic processing system by Advance Geophysical (now Halliburton Landmark)
PVM	Parallel Virtual Machine (software parallelization library)
Qt	Popular cross-platform graphics software library by Trolltech (currently owned by Nokia)
RAID	Redundant Array of Independent Disks
RAYINVR	Popular 2-D wide-angle seismic ray-tracing program by Colin Zelt (Rice University)
RSF	Former name of Madagascar package
SAC	Sesicmic Analysis Code (earthquake data analysis package by Livermore national Lab)
SEGY, SEG-Y	Society of Exploration Geophysicists exchange data file standard
SEP	Stanford Exploration Project

SEP3D	New name of SEPlib package
SEPlib	Stanford Exploration Project software package
SIA	Old name of IGeoS package (1995-2006)
SIOSEIS	Marine reflection package by Scripts Oceanographic Institution
SK	Saskatchewan
SKBG	UofS seismic monitoring station on Bergheim Road Geophysical test site
SKWC	UofS seismic monitoring station at White Cap Dakota First Nation
STA/LTA	Short-time Average/ Long-time average – seismic arrival detection algorithm
SU	Seismic UNIX reflection processing software by Colorado School of Mines, also called Seismic Un*x
TCP/IP	Transmission Control Protocol/Internet Protocol
UofS	University of Saskatchewan
XML	Extensible mark-up language

1 Abstract

IGeoS is an integrated open-source software framework for geophysical data processing under development at the UofS seismology group. Unlike other systems, this processing monitor supports structured multicomponent seismic data streams, multidimensional data traces, and employs a unique backpropagation execution logic. This results in an unusual flexibility of processing, allowing the system to handle nearly any geophysical data.

In this project, a modern and feature-rich Graphical User Interface (GUI) was developed for the system, allowing editing and submission of processing flows and interaction with running jobs. Multiple jobs can be executed in a distributed multi-processor networks and controlled from the same GUI. Jobs, in their turn, can also be parallelized to take advantage of parallel processing environments such as local area networks and Beowulf clusters.

A 3D/2D interactive display server was created and integrated with the IGeoS geophysical data processing framework. With introduction of this major component, the IGeoS system becomes conceptually complete and potentially bridges the gap between the traditional processing and interpretation software.

Finally, in a specialized application, network acquisition and relay components were written allowing IGeoS to be used for real-time applications. The completion of this functionality makes the processing and display capabilities of IGeoS available to multiple streams of seismic data from potentially remote sites. Seismic data can be acquired, transferred to the central server, processed, archived, and events picked and placed in database completely automatically.

2 Introduction

Open-source software has become a significant and integral part of many computing environments. The Linux operating system is perhaps the best known example and it is used in nearly all markets by academics, industry and government. Geophysical software however, has not seen the same level of open-source development and is still dominated by commercial products from a few developers. IGeoS (formerly SIA) has been in development for nearly 14 years as a batch-driven general processing package. It was originally written to replace the commercial package Disco when it was no longer being supported. The project grew into a full featured and robust system for geophysical use but it lacked an interface, integrated display tools, parallel capability, and other modern features that users expect. The goal was not to simply replicate commercial projects but rather to find unique solutions to problems faced in ongoing research.

My work on the project has focused mostly on the display, user interaction, and software updating components. First, a user interface was designed and implemented allowing users to build processing flows, manage projects and access the electronic documentation. Second, a generalized display tool was created to provide visualization and interaction with the data. Finally, a number of other services were added to the package, such as an automated update and code management feature and facilities for transmitting and processing real time seismic data from remote sites.

As a result of the present effort, the whole concept and software implementation of IGeoS were significantly enhanced, transforming it into a versatile and convenient framework for developing software for many geophysical applications. Examples of current applications include 2D and 3D seismic and well-log processing, gravity processing and inversion, earthquake data analysis, 1-, 2-, and 3D seismic waveform modeling, travel-time modeling and inversion, and also continuous seismic monitoring using remote Internet seismographs.

The complete IGeoS code, including all my contributions presented below, is available for downloading, installation, and updates from the main website listed below.

IGeoS home page	www.seis.usask.ca/igeos
Processing web service	www.seis.usask.ca/ps.php
Code maintenance web service	www.seis.usask.ca/cs.php
Module documentation	www.seis.usask.ca/index/index.html
Examples	www.seisweb.usask.ca/temp/examples

General documentation is also provided throughout the website for both users and developers.

2.1 Review of Existing Work

Numerous software projects both commercial and open-source have been created to meet the needs of geophysicists. Yet, there is still a demand within the geophysical community for software that is both easy to use and highly customizable. Commercial software focuses on performing specific tasks within a consistent framework and interface but is not easily adapted to non-standard practices. By contrast, most open-source (geophysical) projects to date lack well-developed interfaces and are not broad enough in their scopes to serve as primary tools for data processing and research.

ProMax is a widely used commercial processing package which specializes in reflection processing. It uses a text based system to build and manage processing flows. While popular, the interface is proprietary and does not support common functions such as dopy/paste, or drag and drop. Further, the program is geared heavily toward processing and includes limited visualization or interpretation tools.

Seismic Unix (SU) is perhaps the best-known open-source geophysical package. It was developed and maintained by the Center for Wave Phenomena (CWP) consortium at the Colorado School of Mines. It contains a large number of tools in a well-documented and maintained package. Each tool is an independent UNIX program that must adhere to a strict input/output structure as the data is passed using file pipes. The tools are arranged by using standard UNIX shell scripts to organize data processing flows. While the package is useful and broadly used, it is still missing an integrated user interface as well as a consistent method for displaying and interacting with data. File pipes, inherently unidirectional, do not easily allow for highly interactive processing flows as there is no method for propagating changes to tools located earlier in the sequence. The file pipe structure also makes it difficult to build a user interface that would be able to interact with the tools concurrently.

Another well-known project is the Stanford Exploration Project library (SEPlib) currently located at http://sepwww.stanford.edu/software/seplib/. It is similar to Seismic Unix in basic design and it was actually the parent project to SU. File pipes are used to pass data between different programs, which each serve as a processing tool. Programs must read and write the specified format to be compatible with other SEPlib programs. Parallel functions are available by using MPI to submit multiple copies of the processing flow across a cluster of computers. SEPlib has been modified to handle irregularly sampled data commonly found in 3D seismic surveys and is now called SEP3D. There are a few graphical display programs and provision to produce plots using a library known as 'vplot.' Vplot, originally written by Dave Hale, allows the scaling and re-sampling of vectors to improve the compatibility between different displays and printers.

Madagascar (formerly RSF) is a recent effort by Sergey Fomel (Personal Communication) to produce a geophysical processing package designed for reproducible computing rather than production processing. Reproducible computing is a response to difficulties that researchers have in reproducing scientific work when complex, multi-stage software is involved. The problems are twofold. First, the exact sequence of processing steps is rarely recorded, and second, new software versions may produce different results. Madagascar (like most packages) is driven by scripts which can be saved to preserve the exact processing sequence. To avoid changes to results from software revisions Madagascar uses a test driven development model. This means that when new code is written, a specification is first developed, and all developments and changes to the code must meet that specification. The design of Madagascar was also inspired by SEPlib.

2.2 Scope of IGeoS

Geophysical processing consists of a sequence of discrete steps organized to produce a result from a set of input data. This sequence is commonly known as a processing flow. Typically, the first tool in the sequence has the job of loading data into the flow. Within the flow the data may be in an internal format, a standard format or just held as array values (IGeoS's method). While data begins with the first tool loading the final tool in the flow usually writes the data out or displays it for the user. Interpretation tools are used to view and interact with the data after processing has occurred.

2.3 Contributions

During the time of my work on IGeoS the package evolved from a processing system to a framework for geophysical code development and interpretation in addition to facilities for real time, remote data monitoring. The design and implementation of the user interface are entirely my effort. It was designed to be familiar to users of other geophysical software and was influenced by packages such as Promax, as well as general software such as kde which also uses the QT libraries. Qt was chosen because of the relative success of the kde environment, the solid commercial grade support and the multi platform nature of the libraries. Additionally, QT fit nicely with the (mostly) C++ code which existed in IGeoS.

I also took responsibility for the design and initial implementation of the visualization components. This included the class hierarchy as well as the mechanism for pushing data and monitoring the processing flow for changes. I also wrote the first version of the OpenGL interface which displayed objects within a QT window.

2.4 Structure of this thesis

This thesis presents the author's contributions to a fairly large and versatile software project (IGeoS) and represents a compilation of the following publications:

Chapter 3 is based on Chubak and Morozov (2006a and b). In this Chapter, I present the recent architecture of the IGeoS package, its goals and relation to other existing system. In particular, I focus on the newly developed Graphical User Interface (GUI), visualization, and parallelization capabilities.

Chapter 4 is based on my contribution to Morozov et al (in press). In this Chapter, I describe the unique interactive, parallel, 3D visualization program that was derived from the GUI above.

In **Chapter 5**, based on Chubak and Morozov (2007), I describe an automated system code maintenance by using distributed web repositories.

Chapter 6 (Chubak et al, 2007a and b) emphasizes the new aspect of the package as a framework for uniform geophysical code development and gives several advanced examples of its application.

Chapter 7, based on Morozov et al. (2007), I present the rationale for seismic monitoring in Saskatchewan and describe a highly automated, low-cost hardware and software solution.   The technical details of its implementation are further developed in **Chapter 8** (Chubak and Morozov, 2008), where a complete system for seismic earthquake monitoring currently operating at the UofS is described. This system is not directly related to IGeoS package; however, it is currently working in close integration with it.

In each Chapter, the text of the original published paper is mostly preserved with the corresponding changes made to section and figure numbering. The reference lists of all publications are collated at the end of this thesis. Some images from the original papers were modified.

Finally, **Chapter 9** summarizes the general conclusions of this work and offers recommendations for further development of this geophysical software project.

3 Integrated software framework for processing of geophysical data

This Chapter is based on publications by Chubak and Morozov (2006a and b). These papers describe the new architecture of the IGeoS package. Since this thesis project started, this architecture has changed substantially from its initial design (Morozov and Smithson, 1997). The SIA package was redesigned using C++ as its core development language, parallelized using the Parallel Virtual Machine, switched to using dynamic linking, and adapted to using Qt and interactive graphics. My development of a modern, Qt-based graphical interface make the project unique among other academic, and even commercial data processing systems. Furthermore, the package was extended to non-seismic applications, such as gravity inversion and modeling and real-time data acquisition. To reflect these changes, the package was recently renamed to IGeoS (for Integrated GeoScience software), under which name it is being continuously developed. These changes led to switching the emphasis from the "wide-angle seismic processing system" (Morozov and Smithson, 1997) to a much broader framework for geophysical data processing. This change of viewpoint became possible largely, due to the contributions from the present thesis, summarized in the Chapter below.

3.1 Introduction

Analysis of geophysical data nearly always involves application of sophisticated and multi-stage processing and inversion. With volumes of data and resolution of the datasets exploding in recent years in nearly every field, the demand for computer packages facilitating handling, processing, analysis, and interpretation of large and complex datasets is growing. This particularly applies to exploration seismology, where development of processing packages has grown into a thriving industry. A number of integrated software systems, mostly specialized and streamlined for reflection seismic data, are available.

Although highly advanced, commercial processing packages are still built for specialized industry users. For a broader geophysical community, reliance on such software may not be satisfactory for several reasons. First, while being highly efficient in their primary fields of application (typically, 2- or 3-D common mid-point reflection data processing), commercial packages could become awkward in handling other types of data. Examples from seismology include wide-aperture reflection-refraction or earthquake data, where native support for flexible, multi-component processing, spherical-Earth geometry, and travel-time analysis is critical. Second, commercial packages often require installation of other systems (e.g., databases or rendering systems) whose support could be difficult or expensive in a University environment. And finally, licensing costs are often prohibitive, particularly when utilizing large multi-processor computer systems.

Open-source seismic processing provides a low-cost alternative to commercial software and, with an appropriately directed development, an ability to adapt to the changing research needs. The best-known examples of such kind are Stanford Exploration Project (SEP) software, SIOSEIS (http://sioseis.ucsd.edu/), and Seismic Un*x, a free reflection processing system developed at the Colorado School of Mines (Stockwell, 1999).

Seismic Un*x has been broadly used in research and teaching seismology (e.g., Templeton & Gough, 1998) and also in smaller-scale seismic processing in industry. However, all these packages are still strongly optimized for reflection processing, and their ability to handle more complex datasets is limited. Examples of such complex datasets commonly encountered in refraction and earthquake seismology include multicomponent, variable-length and sampling interval, seismic records combined travel times and amplitudes. Crustal wide-angle seismology requires an ability to account for the Earth's curvature during data processing and in some cases uses thousands of files for data input. In a broader perspective, a system that could handle borehole logs, potential-field data, velocity and gravity models, and offer improved PostScript rendering capabilities would help to integrate the data analysis and reduce the need for data reformatting.

Here, we present our ongoing development of a system providing the flexibility, and functionality that are found neither in Seismic Un*x nor commercial packages. The system, called SIA (no spelling out available!), was initially developed at the University of Wyoming and continued at the University of Saskatchewan (http://seisweb.usask.ca/SIA). It represents a decade of extensive efforts for integration of academic-style seismic data analysis with the polish and performance of a commercial seismic processor.

The guiding principle of SIA design is decentralization of processing and its abstraction from the content of the character of the particular seismic and geophysical dataset. With the recent modifications of the system, this idea was carried further, to an introduction of dynamic linking, conversion to C++, parallelization, and integration with a Graphical User Interface (GUI). In the following, we describe the development of the package since the previous publications (Morozov and Smithson, 1997; Morozov, 1998). We start with a brief recast of the key design concepts and emphasize the new parallel and GUI functionalities.

3.2 SIA seismic and geophysical data processing system

Initially, SIA started as a replacement for Cogniseis's DISCO processing system to provide a means to use many modules written by the students of the Program for Crustal Studies at the University of Wyoming. Consequently, its key design requirements were typical of massive reflection seismic processing: 1) high throughput achieved by processing tools (modules) operating in a common address space, with custom executables built for each job, 2) seismic processing sequences ("jobs") described using a specialized scripting language and executed in (normally) unattended processes, and 3) a multi-user development and processing environment. In addition, several extensions of the reflection data processing model were made, and particularly, original backpropagation execution logic was introduced (Morozov and Smithson, 1997). The system supported (as it does now) processing scripts similar to those of DISCO.

The structure of an SIA data processing flow, implemented as C++ class named `PROJECT`, is shown schematically in Figure 3.1.

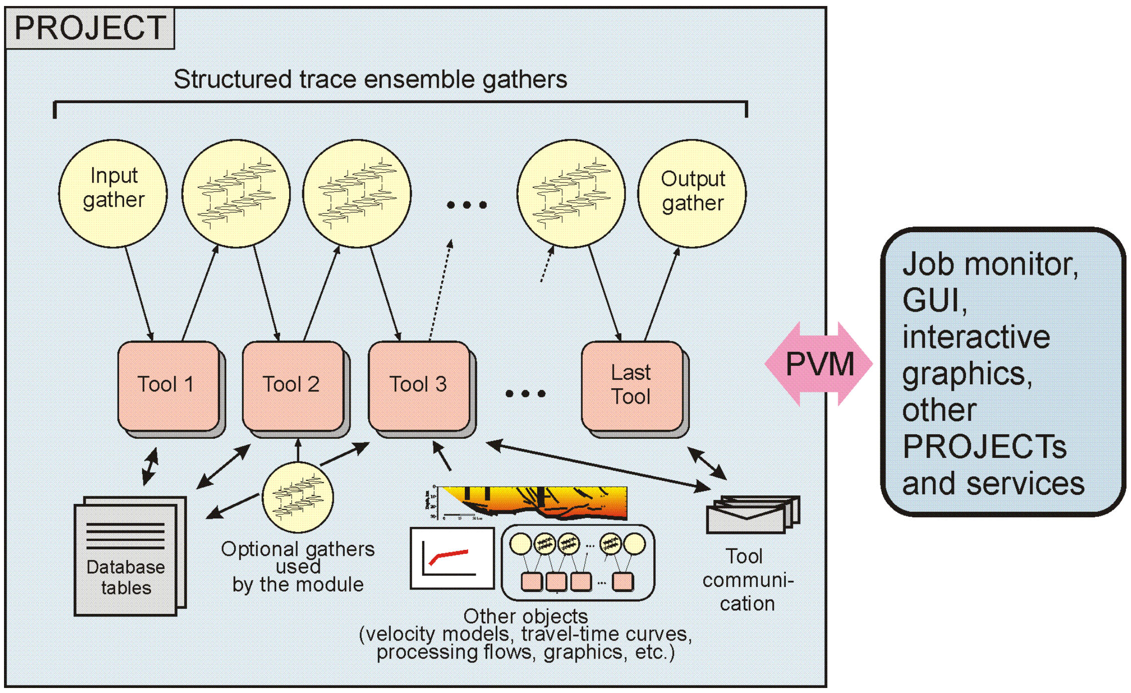

PROJECT
Structured trace ensemble gathers
Input gather
Output gather
Tool 1
Tool 2
Tool 3
Last Tool
Database tables
Optional gathers used by the module
Other objects (velocity models, travel-time curves, processing flows, graphics, etc.)
Tool communi-cation
PVM
Job monitor, GUI, interactive graphics, other PROJECTs and services

Figure 3.1 Structure of SIA processing flow, represented by class `PROJECT` (modified after Morozov and Smithson, 1997). Processing flows consist of linked sequences of tools sharing structured trace ensemble gathers (normally corresponding to multicomponent seismic records), database tables, and various custom data objects. Non-blocking PVM messages are used for communication of flow with GUI, other flows, and services. Note that it has no predefined mechanism for propagating data along the sequence of tools, and hence no dependence on data types.

Several tools are arranged in a sequence according to the processor's requirements. Each of the tools corresponds to a C++ class (referred to as "module" by Morozov and Smithson, 1997) implementing the "Edit Phase" (parameter input) and "Process Phase" (trace processing) methods. These are the only two methods required for seismic processing. Additional methods (such as providing dynamically changing tool names or progress indicators) can be implemented by the modules requiring closer integration with the GUI. The modules can also post their objects (such as velocity models) for their use by other modules in the flow. In addition, the modules have access to globally visible C++ objects providing the SIA system monitor, database, and the Parallel Virtual Machine (PVM) functionalities. All these classes are stored in precompiled object libraries and linked dynamically from shared libraries when the flow is built and started from the GUI or a batch script.

Unlike traditional seismic processing systems (e.g., DISCO, ProMAX, SEPlib, SIOSEIS, or Seismic Un*x) the system has no special input modules and expects no trace data at its input. Some tools (such as performing database operation or plotting) do not perform any operations with the traces, and the corresponding modules do not need to implement the Process Phase. This makes the system more flexible, making it a useful framework for more than just seismic data.

From any module, seismic traces are accessed by contacting the monitor via `SIA.input()` and `SIA.output()` methods. These methods return pointers to the structured input and output trace data gathers shared by the adjacent tools (Figure 3.1; cf. Morozov and Smithson, 1997). The flow monitor takes no part in moving the traces through the tool sequence, and the modules are free to modify the states of both of their inputs and outputs. This could result in data propagation pattern within the flow that could become elaborate (Morozov, 1998); however, for a typical single-trace filtering operation used in most seismic tools, the Process Phase code is quite straightforward:

```
 boolean FILTER::process() {
  TRACE *t = SIA.input()->pass_trace(SIA.output());
  if ( t ) {
    filter(t);
    return OK;
  }
  return FAIL;
}
```

Here, `pass_trace(…)` method transfers the trace to the output of the tool, `return OK` statement informs the monitor that the module has produced an output, and `return FAIL` is used to request more input data (Morozov and Smithson, 1997). The seismic trace is represented by an object of class `TRACE` that provides access to all of its formatting, data, and header information. Trace headers are free-format and are fully customizable by the user (Morozov and Smithson, 1997). The method `filter(TRACE*)` above should implement the desired filtering of trace `t`.

The operation of the monitor is independent of the character of the data being processed and can be briefly summarized as follows (Morozov and Smithson, 1997). When the job is started, all data gathers are emptied and the modules are called recursively in reverse order, starting from the last one to the module currently marked with the “end-of-file” flag. Once a module returns OK (as in the example above), the process is repeated again from the end of the flow. If all modules return `FAIL`, the end-of-file flag is moved to the next (end-of-file) module, and the process is repeated until no modules produce any outputs. This simple scheme resembling the backpropagation inference engine of the programming language PROLOG maintains the minimum possible number of traces in the data buffers and allows the modules to fully control the data flow and termination of the process.

Because the sequence of tool invocations in SIA is driven by a logical inference mechanism rather than by the input data, no restrictions on the types of data or character of processing are imposed. Data can be loaded, removed, or directed backward in the processing sequence, or the flow could operate without input data at all. In the course of its use in several areas of geophysics (mainly wide-aperture, reflection, and teleseismic seismology, and recently 3-D potential fields), data types were considerably generalized and several additional system features were implemented (Figure 3.1):

1) Variable data formats, sampling intervals, record lengths and time starts.
2) "Traces" can now contain linear arrays (seismic records) or 2- and 3-D arrays (multicomponent seismic records, or 2-D grids used in potential field processing).
3) "Tools" can be represented by binary codes or macro-commands combining groups of other tools with coherent parameterization optimized for a particular task, Macro-commands can be defined by any user.
4) Graphics subsystem for rendering complex images in PostScript and building custom Graphical User Interfaces;
5) Extensive use of command line, trace headers, and an integrated job text preprocessor for flexible tool parameterization.
6) Maintenance utilities including automatically generated HTML documentation and tools for generation of macro-commands and processing examples (see http://seisweb.usask.ca/SIA/examples/).
7) Web service allowing execution of complex custom flows on remote systems and providing software updates. This service was developed after the initial version of this paper was submitted and is described in a separate paper (see http://seisweb.usask.ca/SIA/ws.php; Morozov et al., 2007).

3.3 Development

Addition of new tools into the generalized processing framework (Figure 3.1) fills it with the content for the particular area of application. Compared to the original version (Morozov and Smithson, 1997), code development for the system was significantly simplified, mainly due to the use of C++ encapsulation and inheritance, dynamic linking, and improved maintenance and documentation support. The addition of new tools does not require any modification of the monitoring program and can be done by the users. Graduate Geophysics students at the University of Saskatchewan now routinely contribute new tools as parts of their class projects. Code templates are available for the basic methods of data handling, such as one trace in – one trace out, a buffered single gather, or a sliding trace window (see http://seisweb.usask.ca/SIA/examples/templates/).

Although the old code based on the C language (Morozov and Smithson, 1997) is still fully supported, we use the C++ model for all new development. In this model, an SIA tool named, for example, `mytool` is described by a "parameter definition file" `mytool.mpar`. This file contains descriptions of all module's parameters, documentation, C++, C, or Fortran codes or object libraries used for its building. This file is used by a system utility to generate the corresponding UNIX `make` file, resolve library dependencies, and to create both HTML pages and the on-line documentation displayed by the GUI.

Apart from `mytool.mpar`, a single C++ file containing the C function `void *mytool_init()` must also be provided. This function is called once during flow initialization and returns a pointer to the module's data object. Normally, it simply returns `new MYTOOL`, where class `MYTOOL` is derived from a base class `SIA_MODULE` and overloads (if needed) two of its methods:

1) `int MYTOOL::edit()` – the Edit Phase performing parameter input. It returns an integer status specifying whether the module needs to be called during the Process or end-of-file Phases,
2) `boolean MYTOOL::process()`- the Process Phase called when data objects are propagated through the flow, as described above.

To implement the two methods, no knowledge about the monitor operation or presence of other tools is required. Along with `MYTOOL` class, any number of other C/C++, or Fortran codes can be included and placed into the shared module library. Libraries of C and Fortran subroutines and C++ classes (such as performing Fourier transforms, filtering, Least Squares inversion, and implementing complex arithmetic and Matlab-like matrix manipulations) are provided to facilitate development. In our experience, a student familiar with C++ can usually develop a reasonably complex tool in several days,

The configuration of the system allows maintaining multiple versions of the binaries for different computer architectures from a single set of source codes. In such a way, the system was supported at the University of Wyoming and Rice University under Sun Solaris, 32- and 64-bit SGI Irix, and recently under 32- and 64-bit Red Hat Enterprise Linux at the University of Saskatchewan.

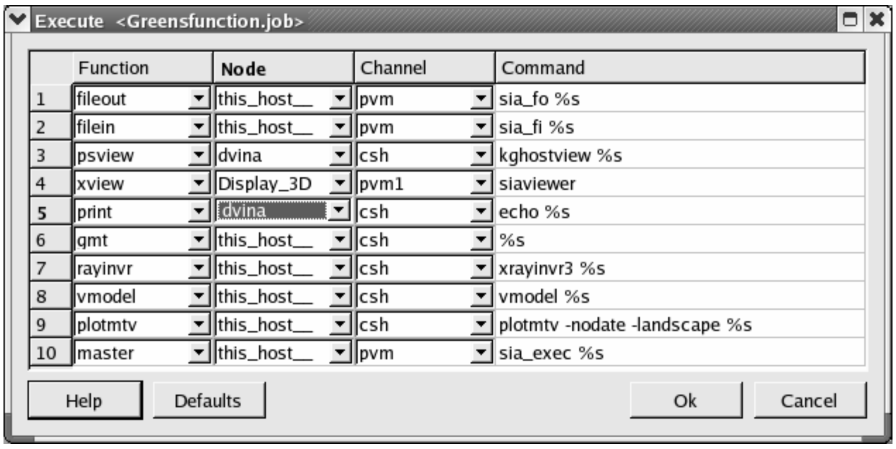
Accumulation and exchange of processing expertise is as important for working on complex research projects as algorithm development, particularly in an educational environment. To date, limited support for systematic documentation is facilitated in SIA by a special tool posting fragments of job scripts in a common database. Any user can select a portion of a processing flow, specify a name and a category for the example, and post it where it can be viewed by others (http://seisweb.usask.ca/SIA/examples). Similar tools create macro-commands and build a library of standard default configurations for the various tools. In addition, processing examples can be simply cut and pasted from, for example, an Internet browser or email.

3.4 Parallelization

The complete processing flow objects can be copied across the PVM interface (Figure 3.1) and executed separate processes on the same or remote hosts. This is the normal mode of GUI operation on multiprocessor subsystems (below), in which all of the computationally-intensive data processing is performed on remote compute servers without overloading the GUI host. Some tools (such as `flow`, used to organize parallel processing; see http://seisweb.usask.ca/SIA/modules/flow/mod.html) spawn groups of processing sub-flows of their own. All processes communicate between each other and with the GUI using `printf(…)` – like messages facilitated by the SIA PVM interface. Along with these messages, the processes also exchange data traces, database tables, and other objects. Execution of the processes is asynchronous, with message queuing and retrieval handled by PVM libraries.

Due to encapsulation of the entire processing in a single `PROJECT` object (Figure 3.1), sub-flows can also be invoked as parts of specialized algorithms. For example, such sub-flows were used to implement custom processing within the loop of generalized pre-stack seismic migration (Morozov and Dueker, 2003).

In order to manage submissions of specialized remote processes, an additional layer of abstraction was created. The user is allowed to define groups of compute hosts and applications assigned to the execution of specific tasks, such as running sub-flows, performing interactive displays, or creating log files. As a result, the tools do not have to specify the exact host and program names but use these task names in order to invoke these applications. For example, depending on the user's definition of "`psview`", a request for a `psview` executes `ghostview`, `kghostview`, `display`, or other PostScript viewing programs on different hosts. In a classroom setting, this technique could provide a near-synchronous cloning of displays on multiple computer screens. Also, parallel jobs can be easily reconfigured for using fewer or more nodes without any changes in their parameters, simply by changing the submission configuration (Figure 3.2).



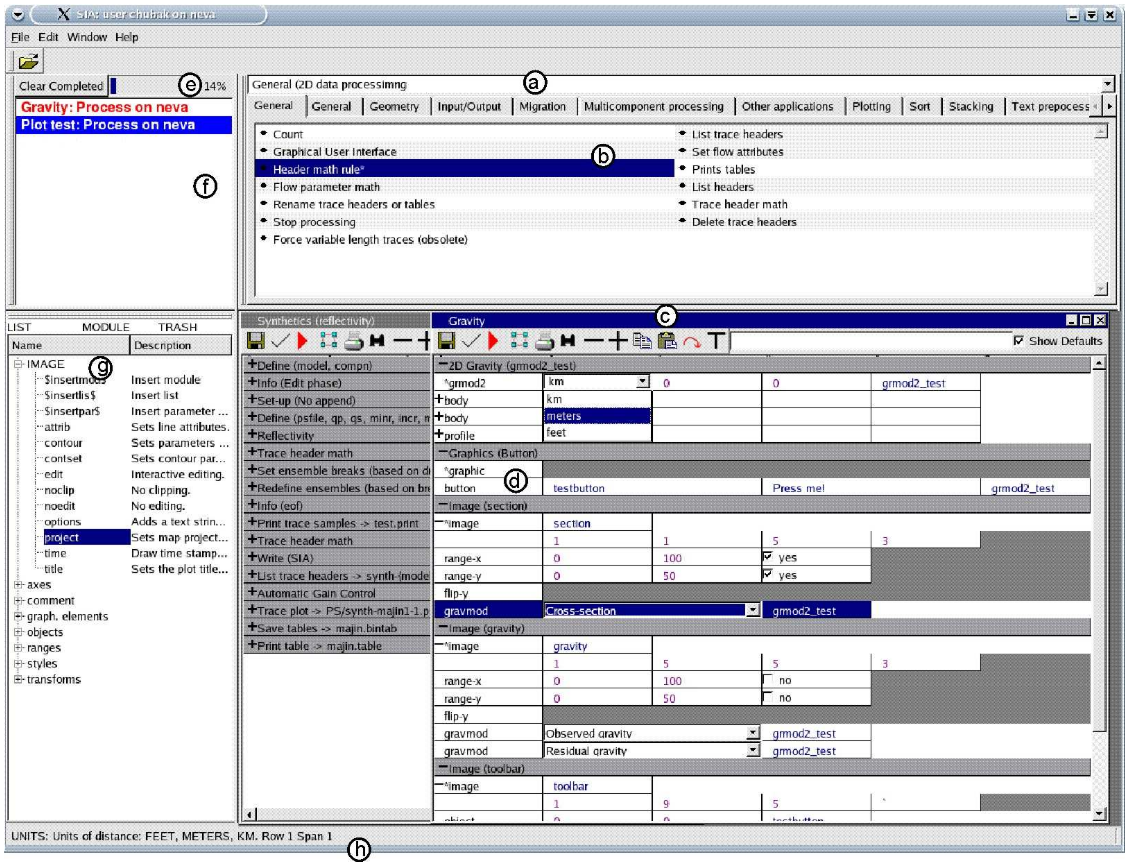
	Function	Node	Channel	Command
1	fileout	this_host__	pvm	sia_fo %s
2	filein	this_host__	pvm	sia_fi %s
3	psview	dvina	csh	kghostview %s
4	xview	Display_3D	pvm1	siaviewer
5	print	dvina	csh	echo %s
6	gmt	this_host__	csh	%s
7	rayinvr	this_host__	csh	xrayinvr3 %s
8	vmodel	this_host__	csh	vmodel %s
9	plotmtv	this_host__	csh	plotmtv -nodate -landscape %s
10	master	this_host__	pvm	sia_exec %s

Figure 3.2 Process submission module of GUI, showing machines available to user via PVM, available task names, types of submission (e.g., via PVM or UNIX shell), and formats of corresponding program calls. Tasks specify symbolic names of actions requested by processing flows (Figure 3.1), such as: "`master`" (for master flow processes), "`compute`" (embedded sub-flows), "`psview`" or "`xview`" (display PostScript or interactive X-windows graphics, respectively), and others. By checking appropriate lines in this list, flow can be executed on different host configurations without changing its parameters. Note that machines listed in this view could represent individual compute hosts or their groups.

Finally, some tools can generate slave processes that do not execute processing flows of the kind shown in Fig. 1.1 yet employ the same PVM communication mechanism. For example, this approach was used to implement 3-D visco-elastic finite-difference modeling integrated into the processing system through module `efd3d` (http://seisweb.usask.ca/SIA/modules/efd3d/mod.html). In this case, model building is performed by broadcasting the corresponding instructions from the Process Phase of `efd3d`, followed by time stepping, editing, and output instructions used to control and synchronize the wavefield simulation.

3.5 Graphical User Interface

As with other similar projects (SEP, Seismic Un*x, SIOSEIS), the advantages of batch (unattended) processing of large volumes of data have historically come at the expense of an intuitive and consistent graphical user interface. Processing jobs had to be described using either UNIX shell or specialized scripts, which always resulted in a significant learning curve and increased the likelihood of errors. A specialized GUI would relieve the processor of scripting, give the system a modern look and feel, and simplify learning by bringing all the documentation to the user's fingertips. Recently, a modern graphical user interface (GUI) was designed for the SIA system (Figure 3.3).



Gravity: Process on neva
Plot test: Process on neva

- Count
- Graphical User Interface
- Header math rule*
- Flow parameter math
- Rename trace headers or tables
- Stop processing
- Force variable length traces (obsolete)
- List trace headers
- Set flow attributes
- Prints tables
- List headers
- Trace header math
- Delete trace headers

Name	Description
IMAGE	
$insertmod$	Insert module
$insertlis$	Insert list
$insertpar$	Insert parameter ...
attrib	Sets line attributes.
contour	Sets parameters ...
contset	Sets contour par...
edit	Interactive editing.
noclip	No clipping.
noedit	No editing.
options	Adds a text strin...
project	Sets map project...
time	Draw time stamp...
title	Sets the plot title...
axes	
comment	
graph. elements	
objects	
ranges	
styles	
transforms	

+Define (model, compn)
+Info (Edit phase)
+Set-up (No append)
+Define (psfile, qp, qs, minr, incr, m
+Reflectivity
+Trace header math
+Set ensemble breaks (based on di
+Redefine ensembles (based on bre
+Info (eof)
+Print trace samples -> test.print
+Trace header math
+Write (SIA)
+List trace headers -> synth-(mode
+Automatic Gain Control
+Trace plot -> PS/synth-majin1-1.p
+Save tables -> majin.bintab
+Print table -> majin.table

−2D Gravity (grmod2_test)					
*grmod2	km	0	0	grmod2_test	
+body	km				
+body	meters				
+profile	feet				
−Graphics (Button)					
*graphic					
button	testbutton		Press me!		grmod2_test
−Image (section)					
−*image	section				
	1	1	5	3	
range-x	0	100	✓ yes		
range-y	0	50	✓ yes		
flip-y					
gravmod	Cross-section		grmod2_test		
−Image (gravity)					
−*image	gravity				
	1	5	5	3	
range-x	0	100	☐ no		
range-y	0	50	☐ no		
flip-y					
gravmod	Observed gravity		grmod2_test		
gravmod	Residual gravity		grmod2_test		
−Image (toolbar)					
−*image	toolbar				
	1	9	5		

UNITS: Units of distance: FEET, METERS, KM. Row 1 Span 1

Figure 3.3 Main SIA Graphical Interface window including: a) selectable tool packages, b) tool library, c) multiple-job flow editor; d) parameterization of selected tool; e) job monitor, and f) status line giving brief information about any item at which cursor is pointing. Job in front of window (c) executes interactive 2-D gravity modeling, and job in back performs 1-D synthetic seismic modeling using *reflectivity* method (Fuchs & Müller, 1971). For a compact display, tool parameterizations can be hidden leaving only one-line summaries that may

change to display job progress (window c). Job flow editor windows (c) also include tool bars providing flow-related functionality.

The GUI is based on the cross-platform Qt libraries from Trolltech (now Nokia), the same libraries on which the popular KDE Linux interface is based. Using Qt relieved us of any X-windows event handling and allowed to incorporate many of the most up-to date GUI design approaches, such as the multiple-document interface, window docking, themes, and platform-independent configuration. Although currently we perform all our development under Linux, other UNIX-type systems such as Solaris, BSD, or Apple's OSX should also work with minimal effort.

The main GUI frame is subdivided into four components that are used most often: the tool library, current module parameters, job editor, and job monitor (Figure 3.3). When the job flow is edited and submitted for execution, the corresponding `PROJECT` object is spawned to the appropriate host(s) and executed. During its operation, `PROJECT` periodically sends information messages to the GUI process, and some of these messages are displayed in the job monitor window. In principle, running jobs may also be programmed to alter some of their parameters which will be immediately displayed by the job editor. At the same time, `PROJECT` is also constantly listening to PVM messages from the GUI, and through these messages, the user can control the remote execution of the flow. Note that different running flows, even those submitted from the same job editor, do not interfere with each other and are independently managed by the monitor.

The **tool library** (Figure 3.3a,b) offers access to over 220 processing tools, about 30 of which are to various degrees experimental. The tools are arranged into packages (e.g., reflection, travel-time, earthquake, potential field data processing, graphics, or development) which may be tailored by the administrators to meet the needs of a variety of users. Within each package, groups of tools (such as input/output, plotting, etc.) are displayed on tab panes (Figure 1.3). A mouse click action on a tool displays its documentation, similar to the one posted at http://seisweb.usask.ca/SIA/sia-index.html. As with ProMAX, typing within the library window invokes a search utility that attempts a keyword search for a tool. When a tool is found, it can be dragged and dropped into the processing flow.

Because the system is intended for users working in different research areas, tools extracted from the different packages could have different pre-set default configurations. For example, applications of the Automatic Gain Control (AGC) in high-resolution, exploration, and earthquake seismology typically use very different time gate lengths. Therefore, we provide several initial configurations for the same AGC tool included in these three packages, and the user is allowed to select the most appropriate configuration.

The job editor (Figure 3.3c) is the central component of the user interface. A multiple-document interface allows several flows to be opened simultaneously, in which the user can edit and execute multiple jobs. Docking windows and tool bars allow a user to customize the layout of the program to make effective use of multi-display systems. Tools and configurations may be copied between jobs, saving the user time and reducing entry errors. Clipboard functions, tool tips, and context-sensitive help are provided to further simplify usage.

Jobs are assigned descriptive names that are passed to the flows during run time (Figure 3.1) and are used to identify them in the job monitor. Job flow descriptions can be built from the tool libraries and examples, and they can also be imported into the interface by dropping text into the flow window. Once placed in the job editor, both tools and parameters can be rearranged by the drag and drop process, making it easy to correct mistakes or change settings.

The job editor displays parameters of all tools in the form of a table (Figure 3.3c). Parameterization can be extended (for example, several hundred lines to describe seismic velocity models). Several types of parameters are currently defined (cf. Morozov and Smithson, 1997): 1) integer, real, double, character string, and Boolean values, 2) selectable and editable text lists, 3) colour, fill, line style, font, and color palette names used by the graphics subsystem, 4) file, module, or flow names, and 5) compute host names, including names of user-defined virtual clusters. Parameters of different types are rendered differently; for example, Boolean values are represented by check boxes, and selectable values – by drop-down lists. Colour highlighting distinguishes between the floating point, integer, and character values.

Proper and sufficient documentation is critical in large-scale processing. At present, processing flow documentation is implemented by allowing the user to attach free-text commentaries to the modules, parameter lists, and parameter groups. The commentaries can be edited and displayed in tool tips.

Once the job parameterization is complete, the flow is submitted for execution through a remote process communication interface utilizing the PVM (Figure 3.1). Jobs may be submitted either for parameter checks (Edit Phase only) or for full processing. If an error is detected or a message issued from a running job, the display of the job (if currently open) is automatically updated using context-dependent colour highlighting. The corresponding error messages are displayed in parameter tool tips and also saved in the job log.

Current module parameterization occupies a permanent window in the GUI (Figure 3.3d) because of its continuous use during editing and also because some of the SIA modules can have quite extensive parameterizations. For example, module `image` (http://seisweb.usask.ca/SIA/modules/image/mod.html) currently offers 57 optional parameter lists to describe its various graphics elements. In the module parameterization window, these lists are displayed graphically in the form of a tree from which the lists can be dragged and dropped into the job editor.

The job monitor (Figure 3.3e) is implemented by simply displaying the information PVM messages received from the running flows and relaying user’s commands back to them. Therefore, operation of job monitor is completely asynchronous and independent from the job editing sessions.

In all GUI components, we make an extensive use of the drag and drop functions, tool tips and status lines to identify options and features while reducing screen clutter. The fonts, colours and other options can be modified to improve their appearance.

3.6 Discussion and conclusions

Although initially designed to extend a reflection processing package (DISCO) to wide-angle seismic data analysis (Morozov and Smithson, 1997), its generalized processing logic have allowed SIA to be extensible to a far broader range of applications. Neither its processing flows nor core databases (Figure 3.1) utilize the specifics of seismic data analysis. The tools are not limited in their types of operation, and a number of non-seismic applications were included into SIA (see http://seisweb.usask.ca/SIA/examples), with the benefits of uniform parameterization, GUI, interaction with other tools, web service, and unified software maintenance and documentation.

The development of the system was driven by the needs of a fairly broad research program extending from shallow to regional and global seismology (http://seisweb.usask.ca/ibm/research.html). As a consequence of this broad scope, SIA offers capabilities for nearly complete reflection and wide-aperture seismic processing combined with support for multicomponent, variable-format data, extensive database capabilities, and input/output in several formats (e.g., SEG-2, SEGY, PASSCAL-SEGY, SEG-P, GSE3.0, CSS3.0, and SAC). Several original inversion codes (such as 2-D reflection and generalized 3-D receiver function migration, genetic algorithms, artificial neural networks, and parallel 1-D and 3-D finite difference modeling) were developed. Tools for 2-D and 3-D processing and inversion of potential fields were recently included. Interfaces to popular program packages, such as Datascope, Generic Mapping Tools (Wessel and Smith, 1995), *rayinvr* (Zelt & Smith, 1992), *reflectivity* (Fuchs & Müller, 1971), and Seismic Un*x, simplify interoperability with other approaches.

Although SIA is being continuously developed, it already represents a fully functional system exceeding its commercial analogs in its scope and many other aspects important for academic researchers. With further development, it could provide an excellent research tool and software development and integration framework for many areas of fundamental and applied geophysics.

3.7 Acknowledgments

The initial SIA development was inspired by one of the authors' (I. M.) research on a number of projects sponsored by NSF (EHR-910-8774-02, EAR99-03235), Gas Research Institute (contract #5089-260-1894) and U.S. Air Force (Grants F49620-94-1-0134, F49620-94-A-0134, F49620-96-1-0326, DSWA01-98-0015, DTRA01-01-C-0057, and DTRA01-01-C-0081). However, none of these projects has directly requested or supported this development.

4 Interactive 3D/2D visualization for geophysical data processing and interpretation

The development of a modern GUI interface (Chapter 3) laid the foundation for building other graphical capabilities of the system. By using the same general concepts, I wrote the initial “xviewer” program that supported an abstract and custom image display protocol. This program was further developed by Shannon Blyth (UofS undergraduate student), and my supervisor (I. Morozov) designed many processing examples and the corresponding client IGeoS tool functionalities utilizing this protocol.   This Chapter describes this protocol and its applications, based on the paper by Morozov et al. (currently in press in Computers and Geosciences).

4.1 Introduction

In several previous publications (Morozov and Smithson, 1997, 1998; Chubak and Morozov, 2006; Morozov et al., 2006), we described development of an open-source software package for geophysical data handling, analysis, and modeling, which we called SIA (http://seisweb.usask.ca/SIA). Started initially as a multicompnent seismic processing package, the approach proved to be quite unique in its broad scope covering the full spectrum of seismic, potential-field, and other geophysical data analysis, but particularly in its implementation including object-oriented design based on C++, dynamic linking, an integrated full-featured Graphical User Interface, parallel functionality, and web services (http://seisweb.usask.ca/SIA/ps.php). Its abstract, logic-based back-propagation data handling model (Morozov and Smithson, 1997), the ease of implementing new tools, high code integration, and extensive documentation and development support allowed extending the system into a code development framework suitable for most tasks encountered in applied geophysics (Chubak and Morozov, 2006). The term back propagation was first applied by Morozov and Smithson (1997) to the IGeoS processing method and the tem is used here to describe the method where a tool in a processing sequence attempts to produce and output and if necessary reads from the tool before it. In this was data is data is effectively drawn from the end rather than pushed from the beginning. Recently, an automatic software distribution and updating service (http://seisweb.usask.ca/SIA/cs.php) was added to the system (Chubak and Morozov, in press), which facilitated concurrent development and automatic maintenance of the package from source code developed by programmers collaborating across the web.

In this paper, we continue presentation of the SIA framework and focus on its new component – general-purpose, customizable, interactive 3D/2D visualization server program. As with other components of the system, in designing this server, we emphasized universality, scalability, efficiency, and parallelism. The resulting code is nearly entirely content-agnostic and suitable for working with most types of geophysical data, in both passive (as data “viewer”) and interactive (“editor”) roles.

Traditionally, geophysical software packages developed, for example, in the reflection seismic industry have been differentiated into “processing” and “interpretation” systems. Processing systems emphasize flow-based design, with numerous operations applied to the data in complex processing sequences, and only limited interactive functionality offered by the individual tools. Special emphasis is made on reproducibility of the results and batch (unattended) execution, often using multi-processor (up to several thousand nodes) computer networks. In the open-source community, several seismic processing systems were developed, such as the Seismic Un*x (Stockwell, 1999). However, these systems still offer only basic user interfaces (essentially, UNIX and Perl shells) and most importantly, are restrictive in their data formats (typically SEGY-like formatted UNIX pipes or files), limited scopes and integration of the tools.

By contrast, interpretation systems are visualization-centred and based on data viewers (for a 3D seismic open-source example see OpendTect, http://www2.opendtect.org/). In such a system, data organization follows spatial patterns, and system operation is mostly driven by data displays and user commands. Application of various "plug-in" tools is typically determined interactively by the user, and only a limited number of fast operations can be performed in real time.

In our visualization approach, we endeavour to erase the above differentiation between processing and interpretation workflows and perform them on a common software base and user interface. As described below, complex images and user interfaces can be defined by the user as parts of SIA data processing flows. These images can then be rendered either in publication-quality PostScript (based on the interfaced GMT programs; Wessel and Smith, 1995) or using the new interactive OpenGL-based display server described here. Because the content of the display is entirely determined by the underlying processing, the display server can implement any functionality, such as displaying seismic data and performing gravity modeling and seismic ray tracing in the same session. Typically the viewer is used to view the output of a tool in the processing sequence and modify processing parameters. General functions such as designing the processing flow, selecting input data, and storing output are handled at the time of flow construction. Through direct access to GMT databases, the server is also able to include 3D coastline base maps in its displays. In addition, full seismic and other data processing capability is also available to the interpreter through the underlying batch flow capability.

Below, we outline the design of the new SIA display server. In a short publication, it is not practical to describe either the features of the program or its code in detail, and therefore, we only emphasize the fundamental, "framework" aspect of the system by focusing on its data abstraction and processing/interpretation model while leaving aside its numerous applications. We begin with the underlying parallel object data communication protocol, followed by a summary of the key features of the software. Further, we explain how complex images are formed and interactivity programmed into the processing flows and present several application examples. In conclusion, we briefly discuss the significance and potential extensions of this approach.

4.2 Object image protocol

Job execution in the SIA system currently incorporates the Graphical User Interface (including processing flow editors, cluster configuration, etc.) and multiple data processing/modeling tasks. These programs operate in independent UNIX processes communicating via a Parallel Virtual Machine (PVM) interface (Chubak and Morozov, 2006; Figure 4.1). Because of some concerns about the continuity of PVM support, its calls are wrapped into a single C++ class that could potentially be replaced with another inter-process communication library in the future.

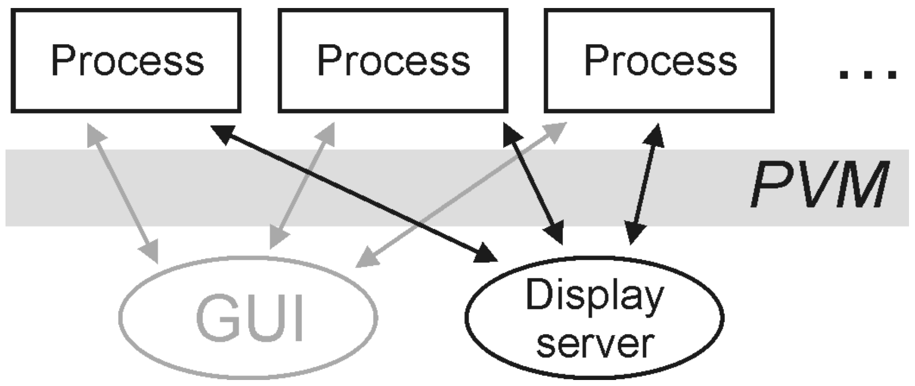


Figure 4.1 Simplified SIA process communication. The Graphical User Interface spawns processing flows (boxes labelled "process") and monitors their execution. If interactive graphics is requested in a flow, it starts a new display server or connects to an already running one. PVM supports two-way communication between the programs which may run on different compute hosts. Grey arrows represent the job monitor PVM messages (flows, data, signals) and black arrows – graphical objects (see text for discussion).

The display server is started by a tool called "gui" included in the processing flows, and similarly to the GUI, it maintains two-way PVM communication with them (Figure 4.1). Processing flows can be configured to execute multiple displays across the network (Chubak and Morozov, 2006); however, for a single user and display host, the same server handles requests from all flows (Figure 4.1). Therefore, if needed, the resulting images can contain objects mapped from different processing flows distributed across the computer network.

Inter-process communication is carried out using asynchronous tagged PVM messages (Figure 4.1). Tags are used for recognition of the message contents, and message formats are automatically converted between different computer architectures by the PVM interface.

The use of PVM messages in the GUI (grey arrows in Figure 4.1) is different from the display server (black arrows). In the GUI, messages are used as instructions controlling program operation by submitting processing flows, passing signals and data, and retrieving results. By contrast, the display server uses PVM messages to maintain hierarchical trees of data objects representing the images being displayed, without interfering with the normal processing sequences. Identical image trees are stored on both the client (flow) and server sides, and data exchange is carried out automatically whenever either of these sides is updated. The communication is thus entirely bi-directional and performed on the background, allowing construction of data displays as well as interactive editing tools.

Schematically, the structure of an image tree is illustrated in Figure 4.2. Each node of the tree represents a C++ object that is able to move across the PVM interface (Figure 4.1). When an update to such an object is received by the display server, it performs the requested action. For example, the "canvas" object initiates a new display window or updates it, "layout" subdivides the window into layout grid, and other objects place their respective images into the grid. Object "image" carries coordinate mapping information, and numerous components of the "graphics" database are not displayed themselves but provide image colours, line and fill styles, palettes, lighting, and other parameters (Figure 4.2A group of objects (buttons, sliders, etc.) provide user controls that can be placed on the image for interactive functionality.

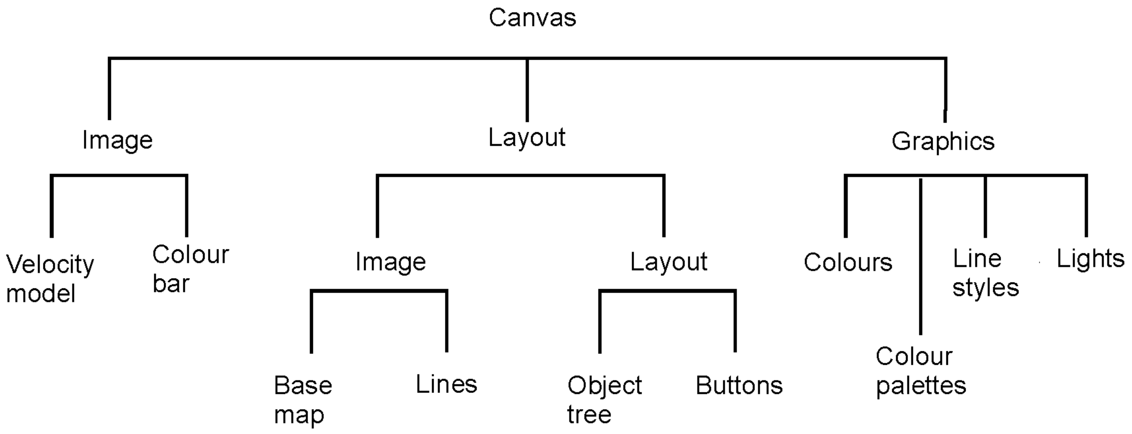


Figure 4.2 Image tree for a hypothetical Graphical User Interface designed by the user. Each node corresponds to an object sent through the PVM communication pipe (black arrows in Figure 4.1). Two copies of such a tree are maintained at both client (flow) and display server sides (Figure 4.1).

Most importantly, the object image trees are included in neither the display server nor processing flow codes. The images are built entirely by the user by placing the appropriate SIA tools into the flows, as described below (examples are given in Appendices A and B). In particular, tools "image" provide most of the general-purpose objects (2D and 3D lines, surfaces, grids), and tool "graphic" introduces line and fill styles, layouts, markers, colour palettes, buttons, etc. In addition, specialized tools provide their own objects, such as 2D velocity and gravity models (tools "rayinvr," "tracer," and "grmod2"), or trace sections (tools "plot" and "plotrt"). In principle, whenever useful, any tool can be equipped with a graphical representation. At the same time, the tools take no part in actually displaying the images, which maintain themselves in the image automatically, as described below. This makes the application code simple and robust.

4.3 SIA 3D display server

The display server is implemented by using Qt C++ graphics libraries (on which, for example, the popular KDE graphical environment under Linux is based), with complex 3D/2D graphics using OpenGL. This ensures that the system will work on a wide variety of systems and takes advantage of the hardware acceleration on graphics cards and processors. It also allows native 3D rendering on stereoscopic displays, such as Geowall (http://geowall.geo.lsa.umich.edu/). The use of C++ ensures the best possible performance and high code integration and reuse. The look and feel of the program is similar to that of typical modern graphical user interfaces (with drag and drop functionality, status lines, tool tips, and elegant window themes available), and code design follows the general style of Qt and OpenGL programming.

The only items in the display server window predefined by its code design are the main window menu along its top and the status bar in the bottom (Figure 4.3). The remaining main part of the window is subdivided by using docking windows and nested Qt layouts. Layouts are named, described in the processing job, and placed into the object trees together with other objects, which allows construction of both simple and complex displays (Figure 4.3). Note that this system of nested, named image frames is similar to the organization of frame sets in HTML. In addition to layouts, docking windows are used to hold service objects, such as the object directory tree displays, property editors, and custom control panels designed by the user. These docking windows also appear only when requested by the processing job, and they can be moved to any position on the screen and collapsed into toolbars during the interactive session.

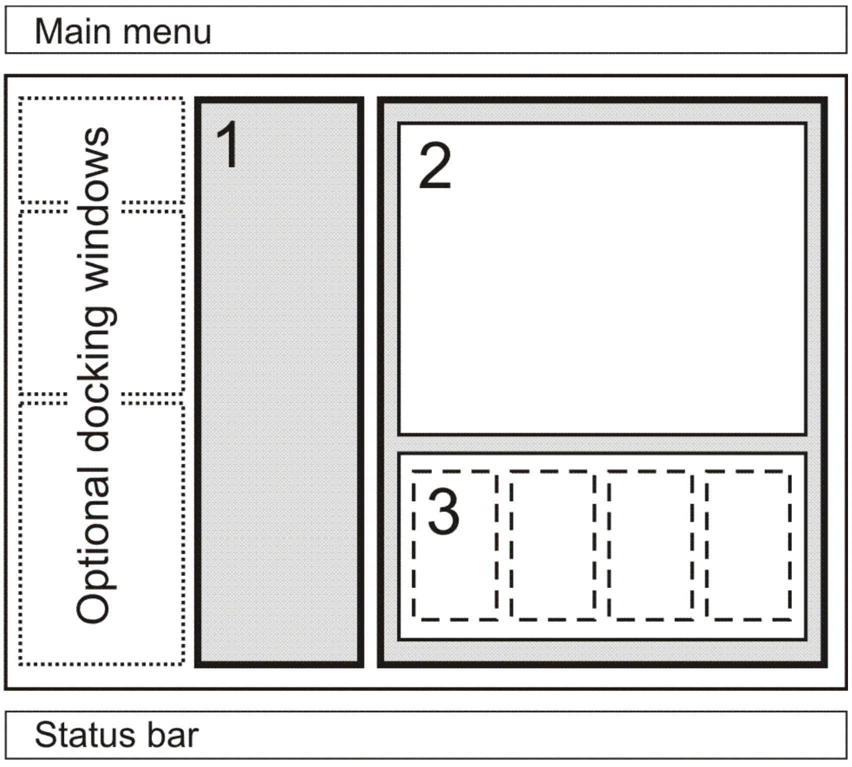
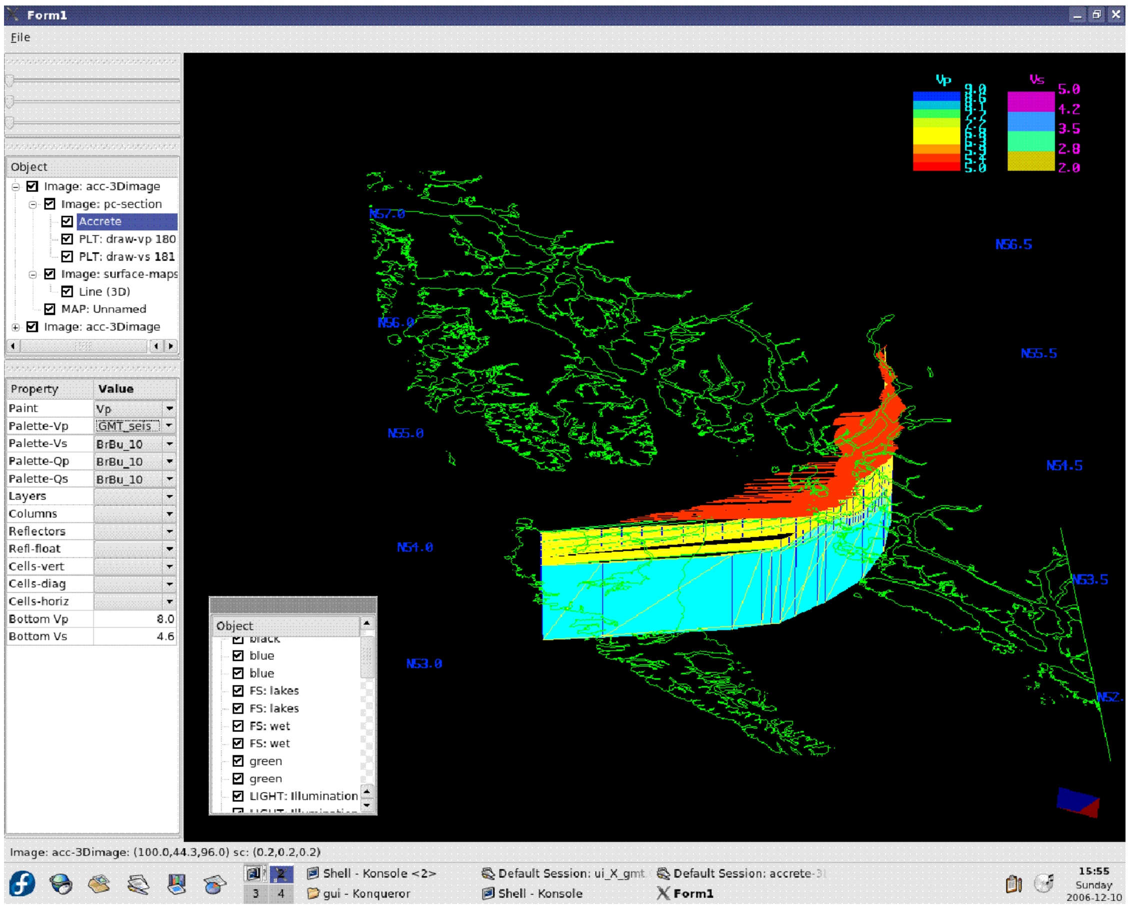


Figure 4.3 Nested layouts for generation of complex co-ordinated displays. Here, the top-level layout 1 is indicated by grey, lower-level layouts 2 and 3 – by white colour and dashed contours, respectively.

The display server program continuously watches its image object tree for updates and rebuilds the display whenever a modification to one of its objects is detected. These modifications can occur as a result of user actions or come from the associated processing flow(s) through the PVM connection. The character of displays is also determined by the objects themselves, and therefore new custom graphical objects can be introduced simply by adding classes into the system library.

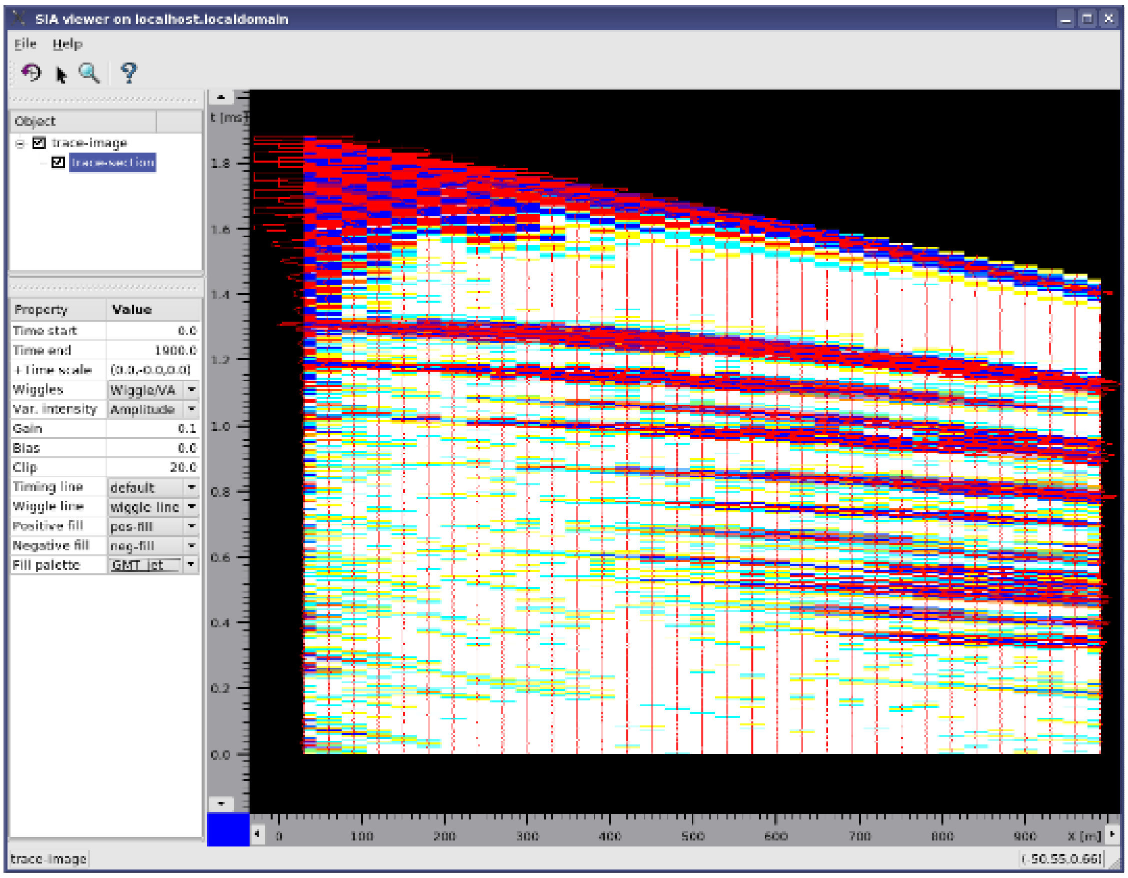
Along with objects representing the imaged geophysical content, the object tree includes auxiliary graphical elements corresponding to Qt-derived classes. Currently, such objects include the object tree viewer, property editor (upper and lower left parts in Figure 4.4-4.6), axes and controls (colour bars, sliders, buttons, and spacers). The object tree and property editor allow the user to interact with the rendered objects and modify them, such as show/hide, change colours, or edit other parameters.



Property	Value
Paint	Vp
Palette-Vp	GMT_seis
Palette-Vs	BrBu_10
Palette-Qp	BrBu_10
Palette-Qs	BrBu_10
Layers	
Columns	
Reflectors	
Refl-float	
Cells-vert	
Cells-diag	
Cells-horiz	
Bottom Vp	8.0
Bottom Vs	4.6

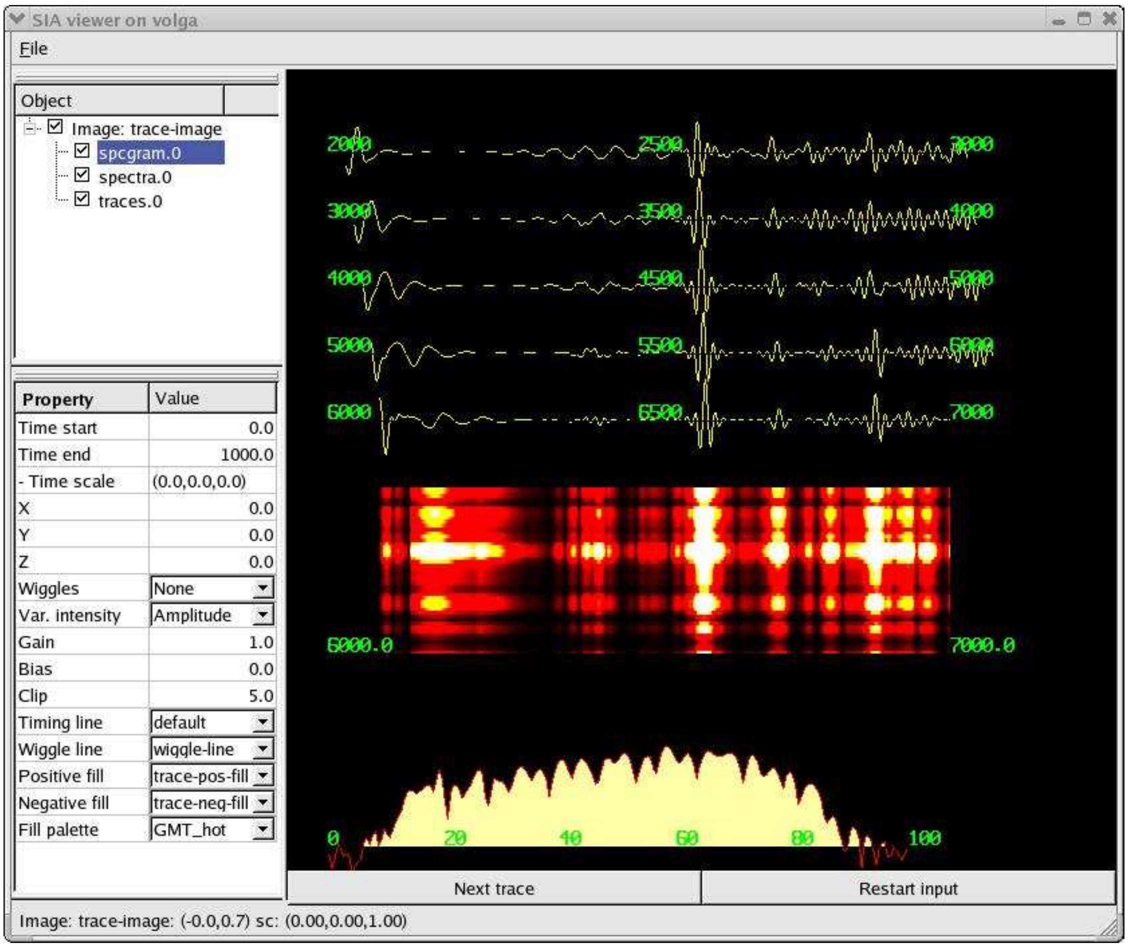
Image: acc-3Dimage: (100.0,44.3,96.0) sc: (0.2,0.2,0.2)

Figure 4.4 Wide-angle crustal ray-tracing model from the ACCRETE wide-angle seismic experiment (Morozov et al., 2001). Property editor allows switching between displays shading using the P- and S-wave velocities, velocity ratios, or wireframe views. Red dots on the surface indicate the source-receiver midpoints. Coastline map is derived directly from GMT databases. Note the editing controls in the Properties menu. Rotation sliders (upper left) can be used for precise rotation around the vertical and horizontal axes. The floating window (inset) summarizes the graphical elements (colours, lines, palettes) that can also be edited in the Property editor. See Appendix A for job files used for this display.



Property	Value
Time start	0.0
Time end	1900.0
+ Time scale	(0.0,-0.0,0.0)
Wiggles	Wiggle/VA
Var. intensity	Amplitude
Gain	0.1
Bias	0.0
Clip	20.0
Timing line	default
Wiggle line	wiggle line
Positive fill	pos-fill
Negative fill	neg-fill
Fill palette	GMT_jet

Figure 4.5 Display of a reflection-refraction shot seismic record. Variable-area wiggle over variable-intensity amplitude display is selected. Note the interactive editing options in the Property editor (lower left). GMT "jet" colour palette (originally re-implemented from Matlab) are used for trace background. Note that over 40 preset palettes are available in this tool, as well as throughout the system.



Object
- Image: trace-image
  - spcgram.0
  - spectra.0
  - traces.0

Property	Value
Time start	0.0
Time end	1000.0
- Time scale	(0.0,0.0,0.0)
X	0.0
Y	0.0
Z	0.0
Wiggles	None
Var. intensity	Amplitude
Gain	1.0
Bias	0.0
Clip	5.0
Timing line	default
Wiggle line	wiggle-line
Positive fill	trace-pos-fill
Negative fill	trace-neg-fill
Fill palette	GMT_hot

Image: trace-image: (-0.0,0.7) sc: (0.00,0.00,1.00)

Figure 4.6 Emulation of the traditional continuous seismic record display. In this SIA job, we load seismic data from files or network interface, subdivide them into segments and display in a scrolling trace sequence (top in the right panel). For the most recent segment, a time-variant spectrogram (middle) and amplitude spectrum (bottom) are also computed and displayed simultaneously. In this example, buttons (bottom) are used to control the data input.

Viewing directions and zoom levels are controlled by the mouse as it is done in other 3D interpretation programs. Optionally, the image aspect ratio can be set to remain constant during screen resizing by modifying the underlying OpenGL transformation matrices. For large objects containing hundreds of thousands of elements, efficiency may become a serous issue. We addressed this issue by automatic resampling the images during rendering, depending on the current viewable area and screen resolution.

An important enhancement of the SIA viewer compared to the traditional 3D displays (e.g., GoCad, OpendTect) is the availability of custom, user-configured "views," such as plan or map views, fence diagrams, frontal cross-sections, projections of 2-D images into 3-D, or preset zooms. These views are created using tool "view3d" and are also represented as named data objects on the image tree (Figure 4.2). In the image property editor, active views can be selected via drop-down menus, allowing quick transitions between them. Figure 4.4 shows an example and Appendix A illustrates the use of this method for creating 3D displays.

Any object on the image tree is allowed to implement an “auto-play” method which is called periodically by a separate thread on the viewer to perform various animations. In particular, auto-play of the image object (Figure 4.2) modifies the OpenGL transformation matrix causing continuous rotation, movement, or zooming the entire image in and out. Such animated displays are sometimes helpful during data interpretation or presentations of the results. As with other options, these playback operations are configured by the user and included in job parameterization for tool “image” (see Appendices A and B).

From a programming standpoint, processing objects (tools) and most graphical objects are introduced into the SIA system by redefining functionality of the base `SIA_MODULE` class. For example, Figure 4.4 shows a 2D visualization using our re-implementation of the popular ray-tracing program *rayinvr* (Zelt and Smith, 1992). The program is interfaced in the SIA package using a tool called *rayinvr*, with its base data class named, by our convention, `RAYINVR`:

```
class RAYINVR : public SIA_MODULE
{
  CHARSTR module_name();      ///< name for the GUI
  int edit();           ///< Edit phase (parameter input)
  boolean process();    ///< Process phase (data processing)
  UI_X *X();            ///< Accessor for the graphical object
  boolean call(…);      ///< custom operation performed by this tool
  …
}
```

Here, method `edit()`defines the parameter input from the job (see examples below and in Appendix A) and `process()`describes the tool functionality during data (in most cases, seismic trace) processing. Note the method `X()`returning a pointer to the object performing graphical representation of the model (Figure 4.4), also derived from the base graphical base class `UI_X`. When “rayinvr” is invoked in the job, this object gets attached to the image tree (Figure 4.2) and propagated to the server (Figure 4.1).

On the display server (where no `RAYINVR` objects are available), an identical graphical object is created by a dynamically-linked subroutine `rayinvr_X()`, also placed on the image tree, and provides all the necessary information for the rendering system. The object builds its images by combining several OpenGL plotting modes (such as sequences of lines, triangles, quadrilaterals, and bitmaps) which are further converted by the display server to OpenGL call lists, optimized, and rendered on the available hardware. In addition to serving the graphics, the object can communicate to its mirror peer in the processing flow, and sometimes to perform data analysis (such as seismic ray tracing in this case).

Note the method `call(…)` in the example above, which is available in many SIA tools and object-tree objects. These methods perform custom data operations requested, for example, by buttons pressed on the user displays. In the case of "rayinvr" tool, these calls perform ray tracing, saving, or exporting the model into files (Figure 4.4). In the following section, we show some examples of how such interactive interfaces are designed.

Finally, the display server supports stereoscopic displays by rendering OpenGL images in two frame buffers using slightly different viewing angles. We have tested this approach on our dual polarized-light projector GeoWall (http://geowall.org) system.

4.4 Custom displays and user interfaces

Interactive 3D or 2D displays are generated by regular SIA processing flows written in a scripting language resembling that of DISCO processing system (Morozov and Smithson, 1997). Job flows can also be created, edited, executed, and saved in XML format when the graphical user environment is used (Chubak and Morozov, 2006). Below, we describe the general scheme of such displays and present several examples from different subject areas, with samples of the corresponding job scripts in Disco-like format shown in Appendix A.

Generally, all SIA displays are created by combining the following key tools in the jobs (see http://seisweb.usask.ca/SIA/index/ and Appendix A):

1) "Graphic" – selects or creates layouts, colours, line and fill styles, colour palettes, buttons, and other objects used in rendering. All these objects are given names (identifiers) by which they can be accessed by other tools;
2) "Image " – creates a sequence of objects from the data content, such as data grids and lines, and attaches them to the image object tree;
3) "Gui" – sends the specified image trees to the display server.

Several instances of "graphic" and "image" tools can be used to create complex displays.

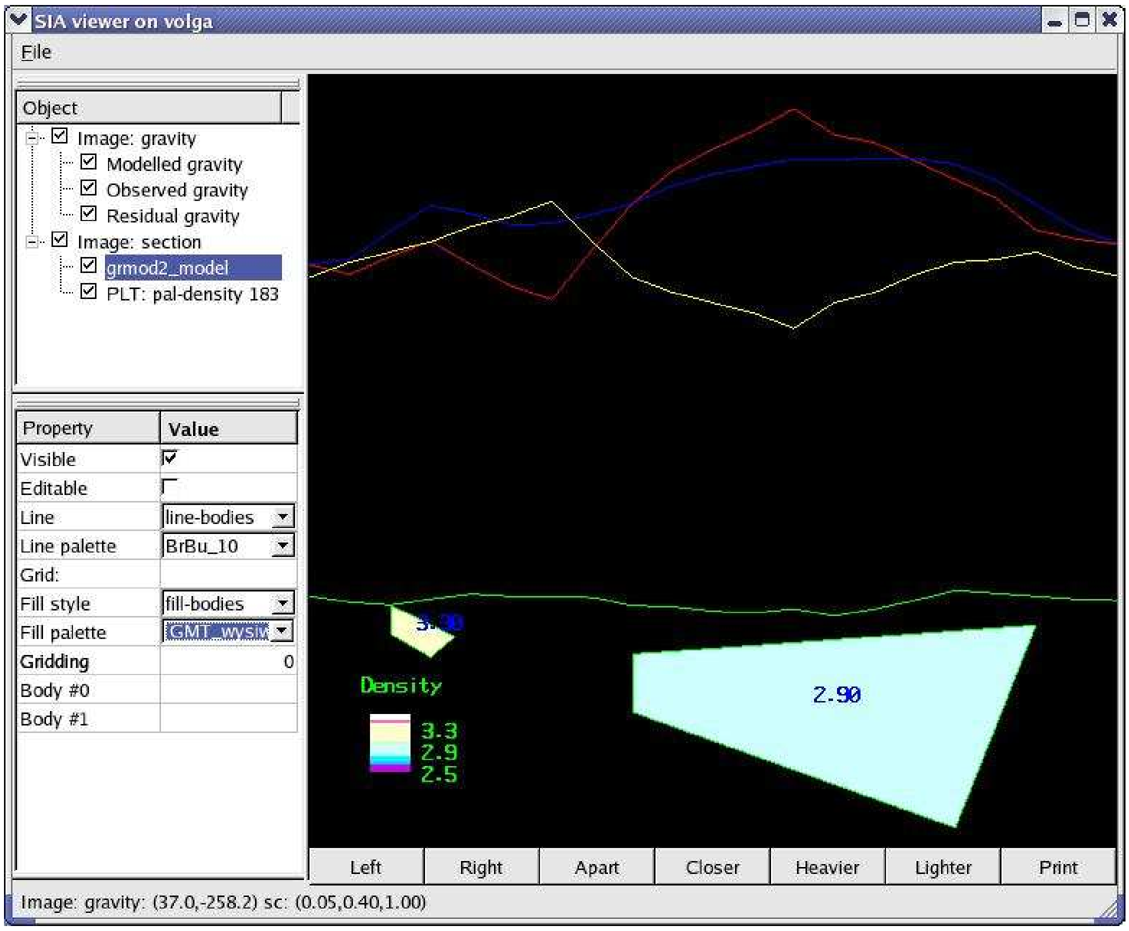
As a first example, interactive ray-tracing and travel-time modeling is perhaps the most important inversion approach employed in wide-angle crustal seismic studies. This procedure requires high-quality interactive graphics, which is limited even in the most popular modeling programs, such as *rayinvr* by Zelt and Smith (1992). Figure 4.4shows an extension of *rayinvr* model in our system, using an example from ACCRETE seismic experiment (Morozov et al., 2001). Along with several enhancements (accurate correction for crooked-line geometry, detailed and non-surface consistent near-surface structure accounting for wide-angle shooting in a fjord, and simultaneous *P*- and *S*-wave ray tracing), the model now allows interactive viewing. Note that the model is created in a two-dimensional (2D) image, which is projected onto a fence diagram in 3D (Appendix A). The model can also be combined with other objects, such as base maps or seismic sections (Figure 4.4). Colour shading can be selected interactively for viewing the *P*- and *S*-wave velocities, velocity ratios, or *Q* (attenuation) parameters, or creating wireframe displays of the model structure. Over 40 preset plus user-defined colour palettes are available, and buttons can be used to perform ray tracing and printing (Figure 4.4). In the near future, the model will also allow interactive editing of its parameters (layer depths and velocities).

As a seismic example, Figure 4.5 shows a synthetic reflection shot gather. The display modes including one-and two-sided variable-area and variable-intensity plotting, trace gain, bias, and clipping, can be adjusted interactively in the Property editor (lower-left part of Figure 4.5). Note that the three-component synthetics were also computed within the SIA package. In another, interactive, example Figure 4.6 shows an implementation of a simple continuous seismic trace display. In the underlying processing flow, the data are loaded from a network connection or continuously updated “ringbuffer” files, filtered and displayed in the form of continuously moving waveforms. Optionally, spectral analysis or event detection algorithms can be included and the corresponding results displayed in the same image (Figure 4.6). In this example, the input is also blocked periodically allowing the user to retrieve one trace at a time by pressing a button on the display (Figure 4.6, Appendix B).

A unique feature of the SIA display server is the availability of coastline data derived from the database files distributed with the Generic Mapping Tools (GMT) package (Wessel and Smith, 1995). Coastline contours (including rivers, channels, state and marine boundaries) and polygons are rendered directly in 3D bypassing the need for map projections. Only a specification of the target map region is required, and the resulting image can be combined with any other objects and viewed interactively in 3D (Figure 4.4). The level of resolution is interactively selectable according to the specifications of the available GMT databases, and line and fill colours are editable from the object Property menu (Figure 4.4and Figure 4.6). Note that the GMT databases are accessed by the display server directly, and PVM link is free from transferring large data volumes, resulting in efficient and fast displays.

4.5 Discussion and further development

The most useful result of the development above could be in the enhancement of the geophysical data analysis by integration of its many components. For example, with the new 3D viewing capability, ray-tracing models (Figure 4.4) from multiple crossing lines can be inverted concurrently in a common display and performed together with gravity modeling and analysis of other data. Because the system is not limited to seismic record-based processing, interactive gravity modeling can be readily incorporated in the same graphical framework (Figure 4.7).



Object
- Image: gravity
  - Modelled gravity
  - Observed gravity
  - Residual gravity
- Image: section
  - grmod2_model
  - PLT: pal-density 183

Property	Value
Visible	✓
Editable	
Line	line-bodies
Line palette	BrBu_10
Grid:	
Fill style	fill-bodies
Fill palette	GMT_wysiw
Gridding	0
Body #0	
Body #1	

Image: gravity: (37.0,-258.2) sc: (0.05,0.40,1.00)

Figure 4.7 Interactive 2D gravity modeling example. Several graphics objects (observed, modelled, and residual gravity profiles, and the density model) are posted by gravity modeling (*grmod2*) tool, and buttons added to illustrate the interactive functionality. Colour palette is used to represent the densities or, optionally, the rock types.

Apart from populating the displays with additional graphics objects, an important line of potential development could be to expand the interpretation-style functionality described in the Introduction. Processing flows can also be sent through the PVM connection and placed on the image tree (Figure 4.2), and therefore they can be associated with the various items in the same display. In such a way, the display could become a data integration hub, with programmable processing flows feeding various types of data into it.

Finally, the development of a user-customizable visualization server advances us to the ultimate goals of the project, which can be summarized, using analogies from the popular geophysical software packages, as follows:

1) Open-source, modular seismic processing pipe similar to the Seismic Un*x but with a significantly broader data model and processing logic;

2) Modern graphical user environment and high-performance, common address space processing similar to ProMax or Disco-Focus;

3) Parallel and distributed processing capability in excess of the above;

4) Interactive 3D visualization similar to GoCad or OpendTect;

5) 2D/3D potential-field data analysis and inversion capability;

6) Geophysical “toolbox” processing versatility and style, ultimately resembling that of Matlab;

7) Built-in access to “academic” GIS data and PostScript plotting, similar to GMT;

8) Remote (Internet) data acquisition, real-time displays, and database capability, similar to Datascope or Antelope (http://brtt.com);

9) Web-service operation (we are aware of no analogs to date);

10) Automatic software distribution and updating from source code and collaborative development.

11) Addition of graphical capabilities to the tools should improve the user experience and benefit most of the areas above.

4.6 Conclusion

A new 3D/2D interactive display server was developed for the SIA geophysical data processing framework (Chubak and Morozov, 2006). The server utilizes Qt and OpenGL graphics libraries, and takes advantage of the object-oriented and nearly content-agnostic design of the core SIA processing system. It operates by creating image object trees that are automatically propagated to the server(s) residing on remote hosts producing complex structured and interactive displays. We show applications of this approach to several areas of geophysics.

With introduction of this last major component, the SIA system becomes conceptually complete and becomes capable for bridging the gap between the traditional processing and interpretation software. Its unusually broad scope includes: 1) high-performance, object-oriented data processing; 2) applications to many types of seismic and non-seismic geophysics; 3) parallel operation on multiprocessor computer networks, 4) processing web services; 5) support for collaboration and automatic software updating; and now 6) parallel, interactive, and animated 3D visualization.

5 Automated maintenance of geophysical software from distributed web repositories

In this short Chapter, based on Chubak and Morozov (2007), I describe an automated system for code maintenance by using distributed web repositories, currently functional in IGeoS system. Such tools are unique in academic software and, to my knowledge, also in the geophysical software industry. The concept that I proposed and implemented was inspired by the examples from open-source Linux software projects. Development of these tools have greatly simplified the maintenance of the package, which is now being operated on several types of computers in our lab, and also downloaded and installed by numerous researchers worldwide.

As in most computationally-intensive disciplines, geophysical data analysis involves numerous algorithms. Large volumes of code have been created, including complex multi-function processing systems, which are particularly well developed in reflection seismology (Stockwell 1999). In most cases, data management, processing, or modeling operations can be subdivided into smaller tasks (e.g., input/output, or some filtering), whose code could be standardized and reused. Ideally, good solutions to problems should be implemented once in a generic fashion so that others could benefit from them. Two critical issues arise in the development of such a general processing system: a) a versatile code integration protocol and a common processing environment suitable for its use in different applications are required, and b) with growing body of software, code maintenance tools are needed. Topic a) above was recently discussed by Chubak and Morozov (2006); in this note, we describe the development of topic b) in our geophysical data processing system.

Within the academic community, the development of computer code is still generally performed in an *ad hoc* manner, without investing significant efforts in software distribution and maintenance. Typical codes are developed by a single group, relatively compact, and can be directly exchanged by the researchers. However, in the more general, complex, and extensively developed packages used by numerous researchers (such as SU and GMT - Stockwell 1999; Wessel and Smith 1999), the need for consistent distribution support is already felt, leading to development of installation web sites and shell scripts.

Complex software packages quickly become difficult to maintain. For example, Seismic Un*x (SU; Stockwell, 1999) consists of several hundred programs that must be installed to use the package. The SIA system (Morozov and Smithson, 1997; Morozov, 1998) includes over 100 modules in the dynamically shared library, over 200 tools written in a variety of languages, and numerous documentation files. Each piece of software may have its own prerequisites (PVM, graphics, third-party software, etc.), compiler options and other configuration issues. Installation and maintenance of such packages represents a significant investment of time and effort from the user. The traditional approach of using a configure script and the make utility to assist the user in the installation could become cumbersome as it is not designed for the diversity of code found in processing packages, nor does it address the need to update only certain code without affecting the entire system. Complex software systems thus require sets of specialized utilities which could automate maintenance and simplify installation, ideally by means of a web-based update service keeping the codes up to date as they are being developed at multiple sites.

Automated code updates are broadly used in modern software (such as Microsoft Windows or Adobe Acrobat). The open-source (particularly Linux) community is addressing the broader needs of updating and maintaining programs by using multiple software repositories. Programs such as `apt`, `yum`, `urpmi`, and `emerge` provide the ability to easily update and install software on several types of Linux systems. Using this model, we have implemented an automatic update and installation tools for the SIA system (Morozov and Smithson, 1997).

SIA represents a major effort for providing a common framework for data management and processing encountered in nearly any field of geophysics. The system is infinitely scalable (the number of processors is limited only by external libraries, hardware, etc.), high data pass-through, capable of extensive seismic, travel-time, and potential-field processing. It includes a feature-rich Graphical User Interface (GUI, Chubak and Morozov, 2006), interfaces to popular academic applications (such as SU and GMT), capabilities for parallel computations, can operate as a web service (Morozov et al., 2006), and the development of a 2-/3-D OpenGL graphics layer is underway. In order to streamline code maintenance and to enable collaborative code development, a set of utilities was added to allow users and developers to effortlessly share their code with the community. These codes are being currently used to synchronise the software versions in our group and also for recent distributions.

The new SIA code maintenance package includes four key utilities:

1) Program `sia-config` provides code customization for the current system. It allows specification of the compilers and their switches for the various phases of building the codes. No specific knowledge is required as most features are handled automatically.

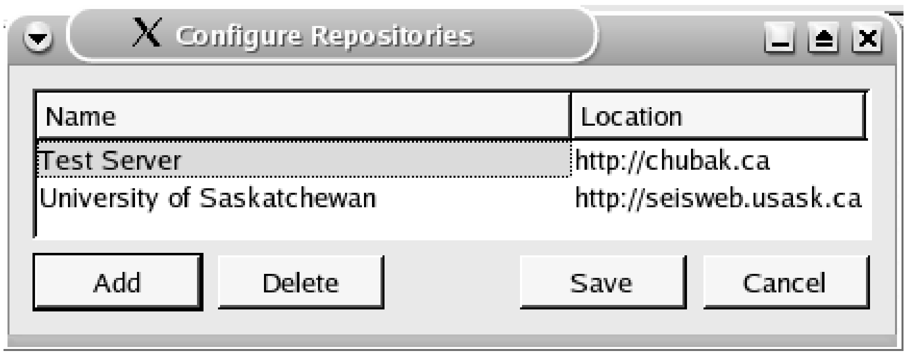
2) Program `sia-update` is the general code maintenance utility performing packing and unpacking of the specified source code components, their building and installation. When installed on a web server, the program also executes most of the code maintenance server requests (`http://seisweb.usask.ca/SIA/cs.php`). On a client, when called with the appropriate switches, the code also tests the resulting binary codes, adds users, generates lists of code repositories and performs file cleanup.

3) Program `sia-install` is the command-line code installer. It obtains the specified components of the package through a web service and installs it by using `sia-update`. For example, command `sia-install http://seisweb.usask.ca –distribution CG .all` installs the "Computers and Geosciences" (CG) subset from our web server.

4) Utility `pdf` registers new plug-in tools with the system (Morozov and Smithson,

1997). This registration includes creating UNIX `make` files, parameter descriptors, GUI menus, and documentation web pages (http://seisweb.usask.ca/SIA//index/).

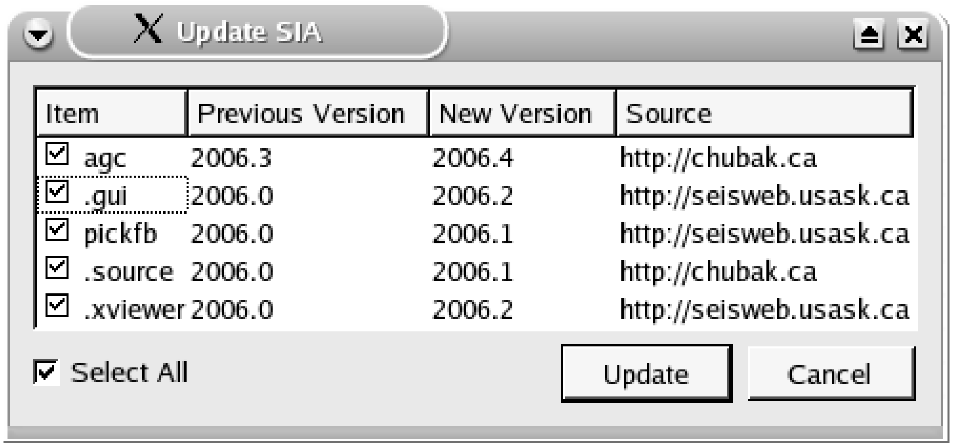
A characteristic feature of our model is that the SIA code is configured identically on all the code repository servers and processing clients. Code servers are therefore also capable of performing full data processing, including remote processing as a web service (`http://seisweb.usask.ca/SIA/ps.php`; Morozov et al., 2006). Conversely, if a standard web server is available on a system used, for example, for specialized data processing and development of the corresponding tools, it can automatically share these tools with others. As a code is updated or added, all clients which connect to this repository will immediately (as soon as the version number is advanced) have access to it. Such symmetrical design makes installation and maintenance of multiple copies of the package easy and reliable.



Name	Location
Test Server	http://chubak.ca
University of Saskatchewan	http://seisweb.usask.ca

Figure 5.1 Configuration of SIA software repositories. Note that only the root entry points are shown as the web addresses. For example, the actual code server for the selected line is `http://chubak.ca/SIA/cs.php`. The buttons below allow the user to edit the list.

In the SIA GUI, users responsible for "administrative" tasks are able to add the URL of any code repositories (Figure 5.1) they are interested in. Upon launch of the GUI, the update client builds a list of locally installed SIA packages including their version information. It then obtains a similar list from each of the servers in the repository list (Figure 5.1). The versions are compared and the user is notified if new packages are available or if updates to already installed packages have been made. As this system is designed primarily to distribute new code no provisions have been made for archiving previous versions. If the user chooses to update or install a software component (Figure 5.2), the source code is downloaded from the appropriate repository and compiled by the `sia-update` utility on the local system. The downloaded code can consist of multiple files in various languages (C++, Fortran, or Java). This utility takes care of all aspects of the installation including generating the `make` files, documentation, and ensuring that the resulting code is optimized for the local architecture.



Update SIA

Item	Previous Version	New Version	Source
☑ agc	2006.3	2006.4	http://chubak.ca
☑ .gui	2006.0	2006.2	http://seisweb.usask.ca
☑ pickfb	2006.0	2006.1	http://seisweb.usask.ca
☑ .source	2006.0	2006.1	http://chubak.ca
☑ .xviewer	2006.0	2006.2	http://seisweb.usask.ca

Figure 5.2 Choosing software components to update. Note that the components whose names begin with a period are system libraries or configuration directories, and the rest are plug-in processing tools (Morozov and Smithson, 1997). For each component, its current and updated version numbers, and the source of the update are displayed. The user can select some or all components which will be downloaded, compiled, and installed.

In conclusion, the ongoing development of the SIA code framework shows that the entire scope of critical issues facing geophysical data management and processing can be solved in a consistent manner. The codes are highly integrated, streamlined for data- or computationally-intensive seismic and non-seismic processing and modeling, make broad provisions for parallelization and remote (web service) operation, and incorporate some of the key community software. With the newly developed web distribution service, the codes can also be developed by multiple authors and seamlessly maintained up to date.

6 Towards a comprehensive open-source system for geophysical data processing and interpretation

Used with permission from the CSEG

The discussion in this Chapter is based on the papers by Chubak et al (2007a and b), in which we emphasized the new aspect of the IGeoS package as a framework for geophysical code development. With such key utilities as the GUI, 3D/2D viewer, and software maintenance system in place, it was shown that practically any type of application (data processing, modeling, inversion, graphics) could be described and efficiently implemented in this framework. The paper below outlined this design philosophy and presented several examples of its application.

6.1 Introduction

Because of its critical importance for modern data acquisition and analysis, geophysical software development has grown into a major industry. Many companies, from majors such as CGG, Landmark, and Schlumberger to numerous smaller vendors provide software solutions and services for numerous applications. Traditionally, geophysical software has been highly specialized for certain applications (e.g., field QC, reflection seismic processing, modeling, or interpretation). However, with growing concentration of computational power, the present and future trends in geophysical software are clearly for re-integration, allowing a researcher instant access to the entire data analysis flow. Another important trend is the explosive growth of open-source software developed and supported by the community.

Our package, called SIA (http://seisweb.usask.ca/SIA) has grown from a diversity of data analysis tasks encountered in an academic environment, and by design, is not limited to any of them. Since its inception in 1995, it was used to process reflection, GPR, and crustal-scale wide-angle seismic data, to create 3D, migrated Receiver Function images of the Earth’s upper mantle, perform travel-time modeling and inversion, process seismic records from nuclear explosions, and recently – to manage a regional seismographic network, to process gravity and air-magnetic images, and even to provide web data services. Started initially as a multi-component interface for CogniSeis DISCO reflection processing package, the approach proved to be quite unique in its broad scope covering the full spectrum of geophysical data analysis.

The open-source model is important for rapid exchange of ideas, development, and response to the needs of the community. The success of open-source software in recent years has demonstrated that it can meet and from many aspects exceed the quality of commercial solutions. With fast development cycles and code contribution directly from users, new features can be quickly implemented and vetted. This has been particularly well demonstrated by the community development centred on the GNU/Linux operating system. The demand for versatile open-source geophysical systems is high – note that in just nine months from November 2006, we received over 190 requests for SIA downloads (Figure 6.1)

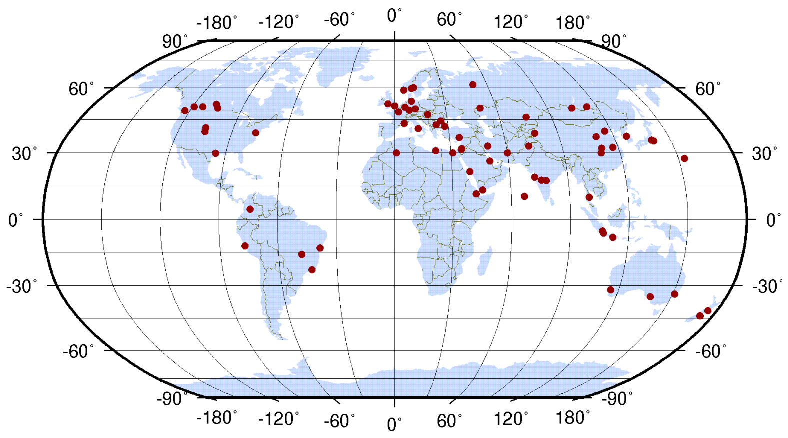


Figure 6.1 Known locations of IGeoS downloads (red dots) from November 2006 to July 2007. Note that the map was produced using the GMT programs (Smith and Wessel, 1995) integrated in the package.

In reflection seismics, many consultants and academics use and write code for Stanford Exploration Project (SEP) and particularly Seismic Un*x (SU) systems because of their maturity and low cost. These systems are adequate for many single-channel applications; however, in more complex tasks, they are strongly limited by linear, UNIX file-stream based design and only basic user interfaces. Matlab (with its free equivalent, Octave) is another popular solution because of its rich toolbox, readily available graphical tools and the ease of developing new processing code. However, Matlab often shows prohibitively poor performance in real data processing problems and requires extensive programming expertise for operation.

SIA system stands out among its counterparts in several respects. It is an open-source solution that endeavours to ultimately provide a comprehensive processing/interpretation solution for the geophysical industry and academia. It provides efficient, dynamically-linked common address space operation (similar to Disco and Promax, and unlike SEP or SU), with significantly richer and customizable data structures and tool interoperability. Its code integration and C++ programming flexibility are similar to those of Matlab. It allows several types of code parallelization and includes libraries and tools for managing multi-processor processing environments. Further, it has a parallel graphical environment with a tightly integrated user interface and customizable 3D data visualization based on cross-platform Qt and OpenGL libraries. Recently, tools for real-time data input and seismographic network management were also added to it. It also has a unique capability of operating remotely, as a web service, and an automatic software distribution and updating service (http://seisweb.usask.ca/SIA/cs.php). These components were described in previous publications (Morozov and Smithson, 1997; Morozov, 1998; Chubak and Morozov, 2006; Morozov et al., 2006; Morozov et al., in review; Morozov et al., 2007). In this paper, we overview the key features that may be of most interest to geophysicists and software industry.

6.2 (Not only) Seismic processing system

SIA currently is a nearly complete seismic processing system, with many tools reaching to the broader geophysical applications (Chubak and Morozov, 2006). The current system scope includes reflection, wide-angle, and to certain degrees earthquake seismology, 2- and 3D potential field processing and inversion, PostScript and interactive graphics. Nearly 200 dynamically-linked plug-in tools are closely integrated with a content-agnostic processing monitor and often between each other forming sub-packages, such as graphics, AVO, or Artificial Neural Networks. Almost any type of data can be handled by the system making it possible to merge multiple data types.

The system was originally a replacement of Disco reflection seismic processing system, and it still supports Disco-style job scripts, with several extensions (see job examples at http://seisweb.usask.ca/temp/examples). Tools written for Disco can also be incorporated, with virtually no modifications.

The key components of the system (the GUI, processing flows, visualization and display tools) operate asynchronously and communicate through a Parallel Virtual Machine (PVM) interface (Figure 3.1). Because of the use of PVM, the many components of SIA can be distributed, allowing, for example, to distribute the processing load or for the visualization program to operate on one or several dedicated computer systems.

6.3 Processing concept

The central concept of SIA is the abstract “processing flow” representing a logical sequence of data manipulation or modeling steps performed by “tools” connected by structured “trace” data buffers (Figure 3.1). The sequence is recursively invoked in reverse order, more resembling the mechanism of logical inference than data processing (Morozov and Smithson, 1997). First, an output is requested from the last tool, if it requires data from the previous tool that tool then attempts to produce it. The processes continues backwards through tools until there is no longer any data available. Note that the flow contains no mechanism for data propagation (this is done entirely by the tools), and thus no assumptions about the data types or character of processing is made. As an example, the system can take a random walk through a 3D seismic dataset (Morozov, 1998). Flows, as well as data traces and many other objects can be transmitted across the PVM connections (Figure 3.1) to potentially form a complex, parallel processing environment.

On top of this abstract processing model, the following features further enhance the flexibility of the system:

“Trace records” (Figure 3.1) can be of variable data formats, sampling intervals, record lengths, and time starts. They can contain linear arrays (seismic records) or 2- and 3D arrays representing multi-component seismic records, or 2D grids used in potential-field processing. However, traces are not required in order for the system to operate.

Other types of data are broadly used and often introduced by new tools (Figure 3.1): velocity models, travel-time curves, database tables, Artificial Neural Networks, inversion engines, and various graphical objects.

User-defined "trace headers" can contain variables of any types (as in Disco or ProMAX, and again unlike SU), but also arrays, references to databases, and functions allowing, for example, "on the fly" computation of midpoints and azimuths based on the endpoint coordinates.

"Tool" parameterization is unusually flexible and uses trace headers, database fields, symbolic text substitutions, and UNIX command-line parameters interchangeably with constants. Many tools support structured parameterizations allowing, for example, to design custom graphical user interfaces (GUIs) or build composite PostScript plots. Tools can be represented by binary codes or macro-commands combining other tools, with coherent parameterization and optimized for a particular task.

Some tools may not participate in the flow (Figure 3.1) at all but instead provide services to other tools. For example, the AVO tool can compute Zoeppritz reflection coefficients or Elastic Impedances for plotting, by using models generated by the tools producing waveform synthetics.

In all geometry manipulations, the system is aware of the Earth's shape, with several ellipsoidal approximations or Cartesian coordinates to choose from.

All processing flows can operate from the user's GUI or from parameterized batch scripts allowing execution of complex, unattended, self-documented processing sequences.

6.4 Graphical User Interface

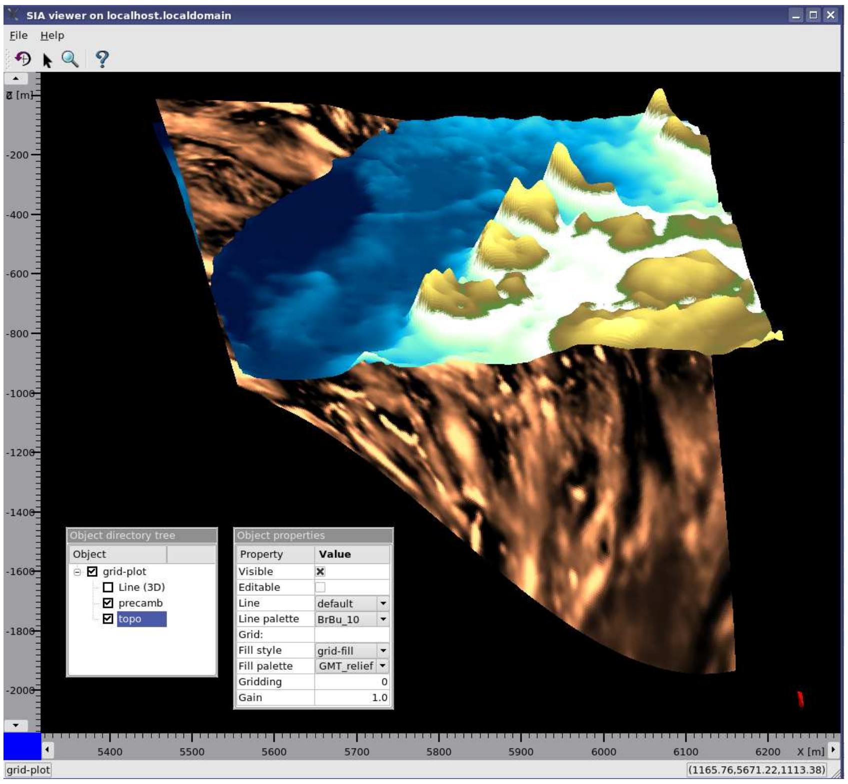
Constructing processing flows is greatly simplified by a modern GUI which also provides the utilities users expect from commercial software, such as project management, process monitoring and control, search, and extensive context-sensitive help (Figure 3.3). The GUI is based on the cross-platform C++ Qt libraries from Trolltech, so that SIA can be ported to a variety of operating systems, such as Linux, Solaris, or even OS X with only minimal effort. In a grid or cluster environment, its configuration is also done from within the GUI by specifying the nodes on which a particular flow and its components (subflows, I/O, display tools) is to be run. This allows multiple processing jobs to be run in parallel on either a Beowulf cluster or distributed over a peer network.

Tool names in the GUI may be context-dependent and showing summaries of their parameters (Figure 3.3). Tools can also communicate the changes in their parameters during run time (e.g., from interactive editing by the user), which would be displayed and saved on closing the job.

6.5 OpenGL/Qt 3D/2D display server

Visualization and interaction with the data is a key to many data analysis tasks. Traditionally, geophysical software packages have been differentiated into "processing" and "interpretation" systems by the role of interactive visualization in them. Processing systems emphasize flow-based design (Figure 3.1), with special emphasis on reproducibility of the results and batch (unattended) execution. By contrast, interpretation systems are visualization-centred and based on data viewers (such as OpendTect, http://www2.opendtect.org/). In such a system, the data organization follows spatial patterns, and system operation is mostly driven by data displays and user commands. Application of various "plug-in" tools is typically determined interactively by the user, and only a limited number of fast operations can be performed in real time.

In our visualization approach, we endeavour to erase the above differentiation between processing and interpretation workflows and perform them on a common software base and user interface. Some examples are shown in Figure 6.2-6.6. By combining tools from the graphics package, complex images and user interfaces can be defined by the user as parts of SIA data processing flows. These images can then be rendered either in publication-quality PostScript (using the interfaced GMT programs; Wessel and Smith, 1995) or by using an interactive OpenGL-based SIA display server. Because the content of the display is entirely determined by the underlying processing, the display server can implement any functionality, such as displaying seismic data and performing potential-field modeling and visualization (Figure 6.2), seismic ray tracing (Figures 2.4 and 4.3), and computing waveform synthetics (Figure 6.4) in the same session. Through direct access to GMT databases, the server is also able to include 3D coastline base maps in its displays (Figure 6.4). In addition, full seismic and other data processing capability is also available to the interpreter through the underlying flow-processing capability.



Object directory tree

Object
☑ grid-plot
☐ Line (3D)
☑ precamb
☑ topo

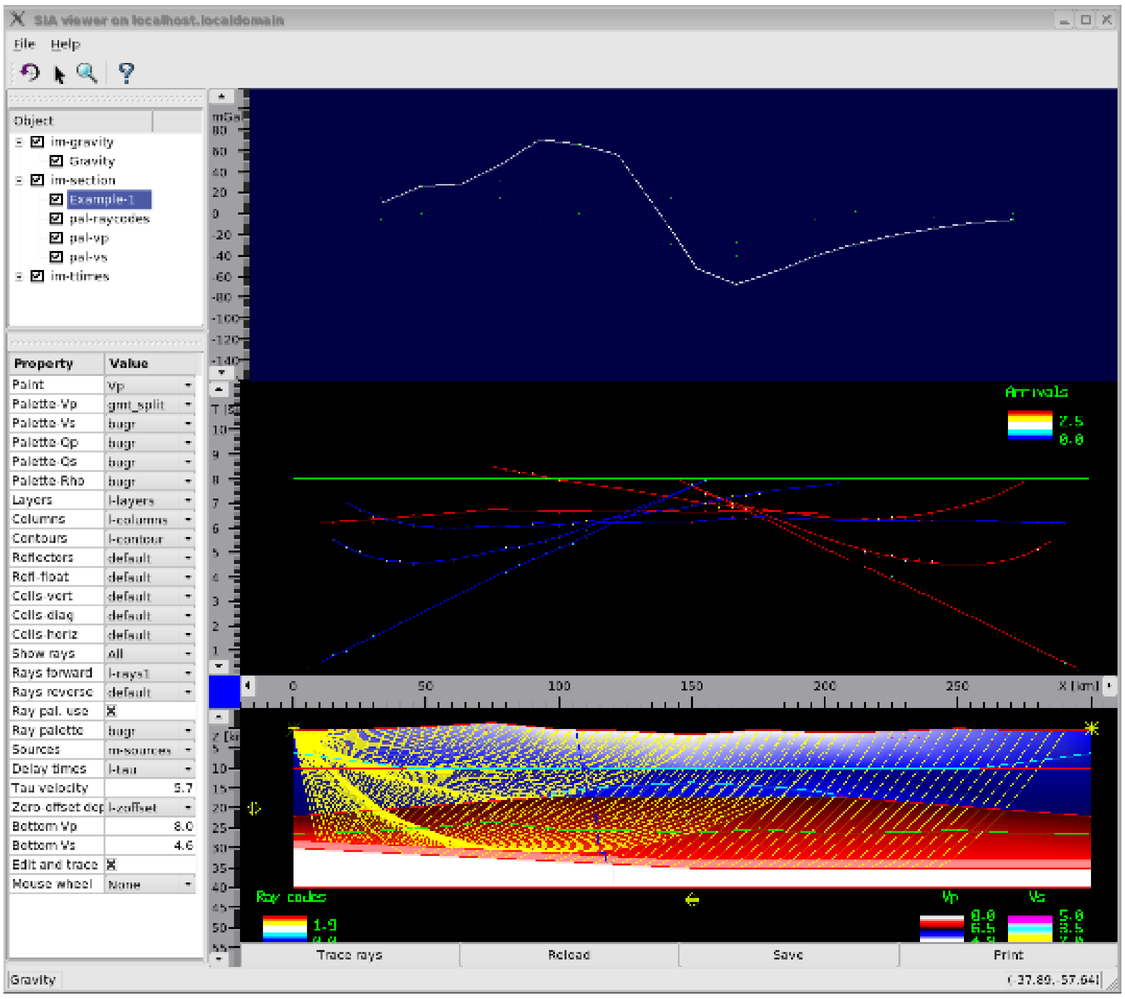
Object properties

Property	Value
Visible	☒
Editable	☐
Line	default
Line palette	BrBu_10
Grid:	
Fill style	grid-fill
Fill palette	GMT_relief
Gridding	0
Gain	1.0

Figure 6.2 An example of interactive 3D visualization for potential-field interpretation. The model shows the Precambrian basement in SE Saskatchewan coloured by air-magnetic anomaly (copper colouring). The surface topography is highlighted using the "sea-land" colour

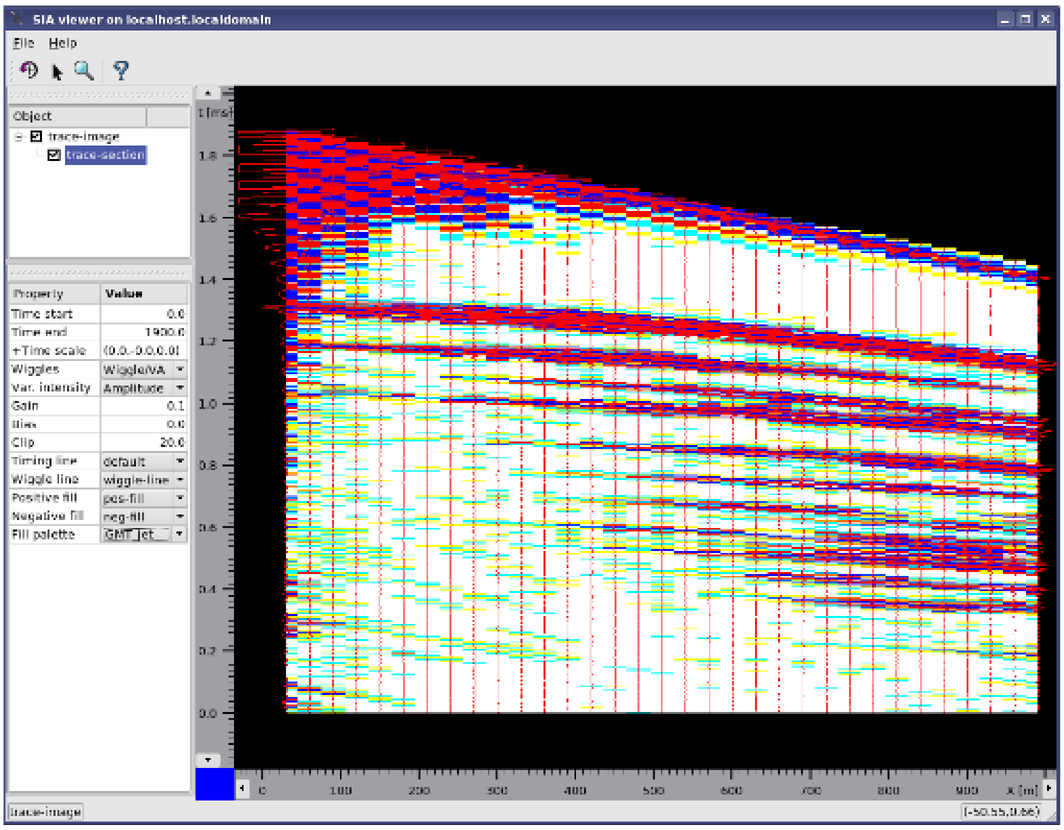
palette from GMT. Over 40 preset colour palettes are available, and custom palettes (as well as colours, line styles, and lighting) can also be defined`

..



Property	Value
Paint	Vp
Palette Vp	gmt_split
Palette Vs	bugr
Palette Qp	bugr
Palette Qs	bugr
Palette Rho	bugr
Layers	l-layers
Columns	l-columns
Contours	l-contour
Reflectors	default
Refl-float	default
Cells-vert	default
Cells-diag	default
Cells-horiz	default
Show rays	All
Rays forward	l-rays1
Rays reverse	default
Ray pal. use	☒
Ray palette	bugr
Sources	m-sources
Delay times	l-tau
Tau velocity	5.7
Zero-offset dep	l-zoffset
Bottom Vp	8.0
Bottom Vs	4.6
Edit and trace	☒
Mouse wheel	None

Figure 6.3 Wide-angle crustal ray-tracing model from the ACCRETE wide-angle seismic experiment (Morozov et al., 2001).



Property	Value
Time start	0.0
Time end	1900.0
+Time scale	(0.0,-0.0,0.0)
Wiggles	Wiggle/VA
Var. intensity	Amplitude
Gain	0.1
Bias	0.0
Clip	20.0
Timing line	default
Wiggle line	wiggle-line
Positive fill	pos-fill
Negative fill	neg-fill
Fill palette	GMT jet

Figure 6.4 Display of a reflection**-refraction shot seismic record. Variable-area wiggle over variable-intensity amplitude display is selected. Note the interactive plotting options in the Property editor (lower left). GMT "jet" colour palette (re-implemented from Matlab) is used for trace background.**

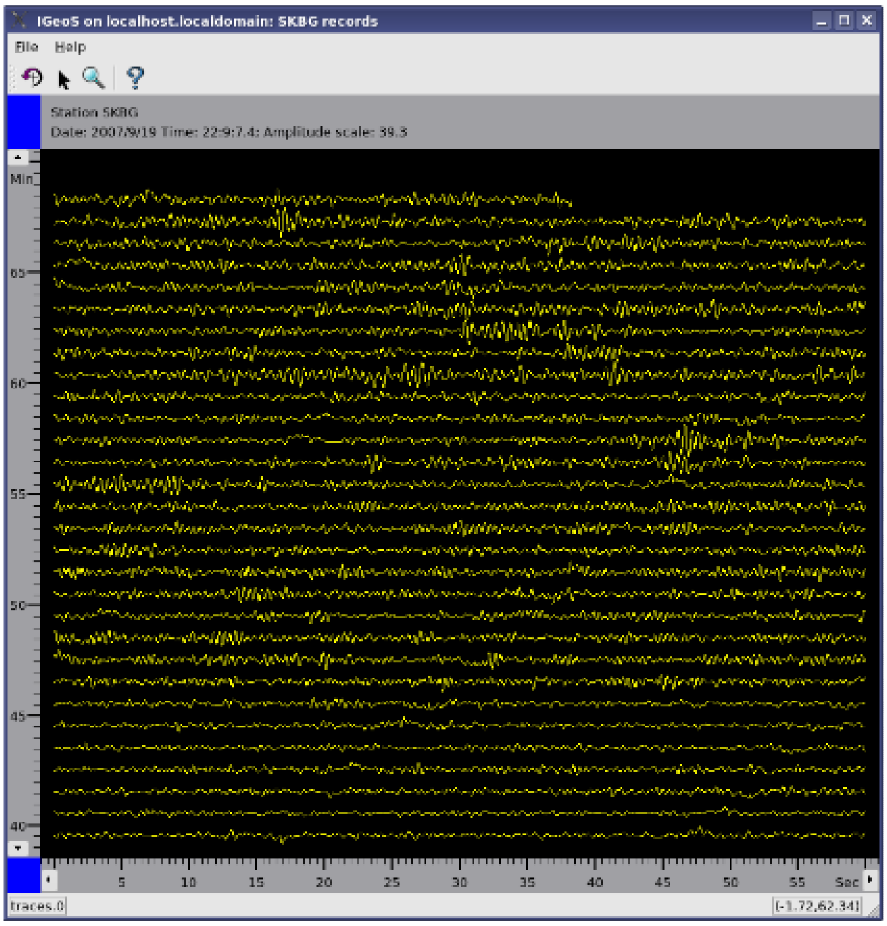
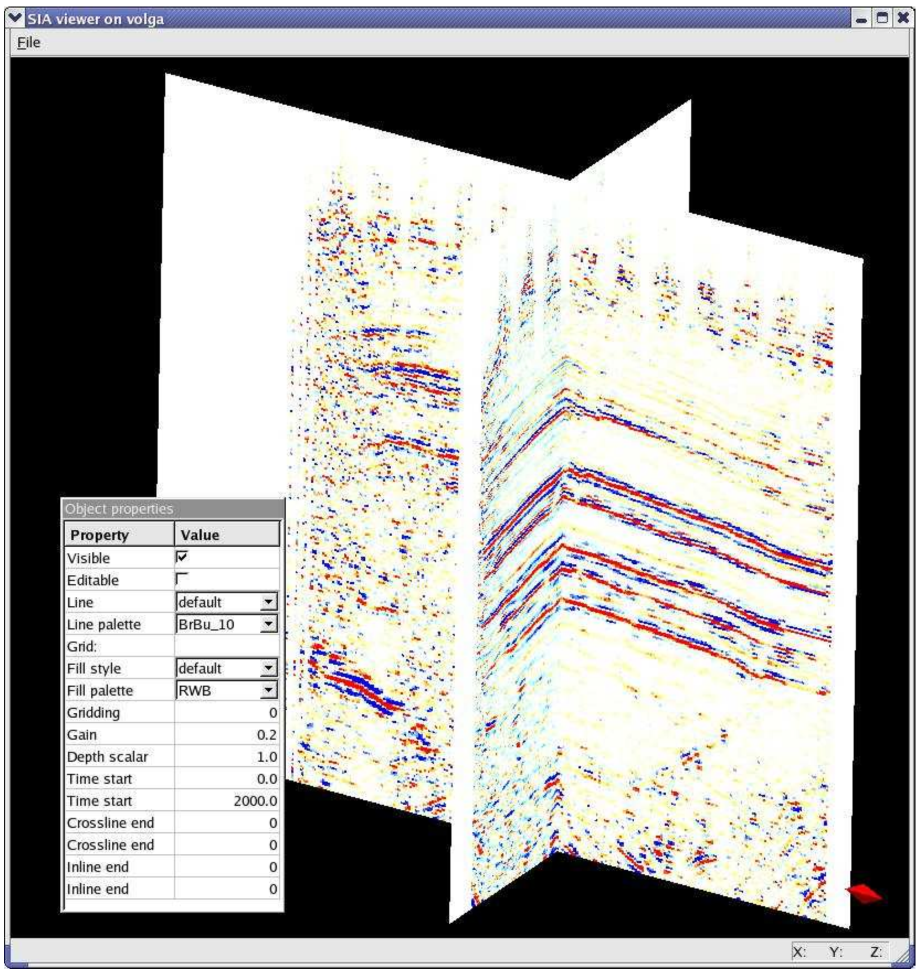

IGeoS on localhost.localdomain: SKBG records
File Help
Station SKBG
Date: 2007/9/19 Time: 22:9:7.4: Amplitude scale: 39.3
Min
65
60
55
50
45
40
5 10 15 20 25 30 35 40 45 50 55 Sec
traces.0
[-1.72,62.34]

Figure 6.5 Continuous seismic record display. In this SIA job, we load seismic data from files or network interface, subdivide them into segments and display in a scrolling trace sequence.



Object properties

Property	Value
Visible	☑
Editable	☐
Line	default
Line palette	BrBu_10
Grid:	
Fill style	default
Fill palette	RWB
Gridding	0
Gain	0.2
Depth scalar	1.0
Time start	0.0
Time start	2000.0
Crossline end	0
Crossline end	0
Inline end	0
Inline end	0

Figure 6.6 3D seismic trace display with a floating object property window.

The visualization system is entirely controlled by the processing flows and is able to render a variety of basic data types including:

Seismic traces with adjustable settings and arbitrarily positioned in 3D (Figure 6.4-6.6).

Lines and surfaces with variable styles, colours, markers, etc.

Bitmap-style graphics rendered on any plane oriented in 3D.

Customizable line styles, colours, colour palettes, axes bars, labels, push-buttons, sliders, etc. (Figure 6.5).

Complex objects (such as velocity and gravity models) are composed of the objects above by the corresponding tools. In addition, user-specified coordinate transformations are available, so that images can be rendered on arbitrary surfaces. This allows, for example, drawing 3D seismic fence diagrams or various displays on the topographic relief or on the surface of ellipsoidal Earth.

The display server operates in parallel on the same or different (optionally, multiple) computer hosts. While interacting with the user, the server also communicates with its master processing flow, causing it to take the appropriate actions. For example, Figure 6.5 shows an implementation of real-time network data input control for a remote Internet seismograph (Morozov et al., 2007). The buttons (bottom of Figure 6.5) are used to control the data input by the master flow performing the seismic network monitoring.

6.6 Integration with popular open-source software

Open-source and open data format design encourages mutual software integration. Several popular academic applications proved to be particularly useful in our work, and they were integrated with the SIA system using specialized tools:

The **Seismic Un*x (SU)** (http://www.cwp.mines.edu/cwpcodes/) is a free and complete seismic reflection processing system broadly used at the academia and by consultants in the industry. It was incorporated virtually entirely by means of SIA tools allowing running SU processing pipes in (remote) parallel processes and exchanging the seismic traces via PVM connections. In addition, several SU codes were "wrapped" into SIA I/O interfaces making them fully compliant with the system. In both cases, the SU tools gained the advantage of the GUI, extended graphics, a more powerful user interface, parallel processing capability, and code maintenance services.

The classic **reflectivity** (propagator matrix) approach for modeling elastic wavefields in 1D, layered models was included in both K. J. Sandmeier's (Fuchs and Muller, 1971) and Kennet's (1993) implementations. These tools are important parts of the emerging AVO package. Both tools have identical model descriptions and output 3-component synthetic seismic traces directly into the job flows (Figure 6.4). The first of these programs was also parallelized for operating on a Beowulf cluster, and is also capable of plotting the models and tracing travel times in them.

3D, parallel, visco-elastic finite-difference modeling (Bohlen, 2002) was revised for encapsulated PVM inter-process communication and integrated with the GUI. Currently, work is underway for providing an accurate topographic free-surface condition, 3D model visualization, and interactive model building.

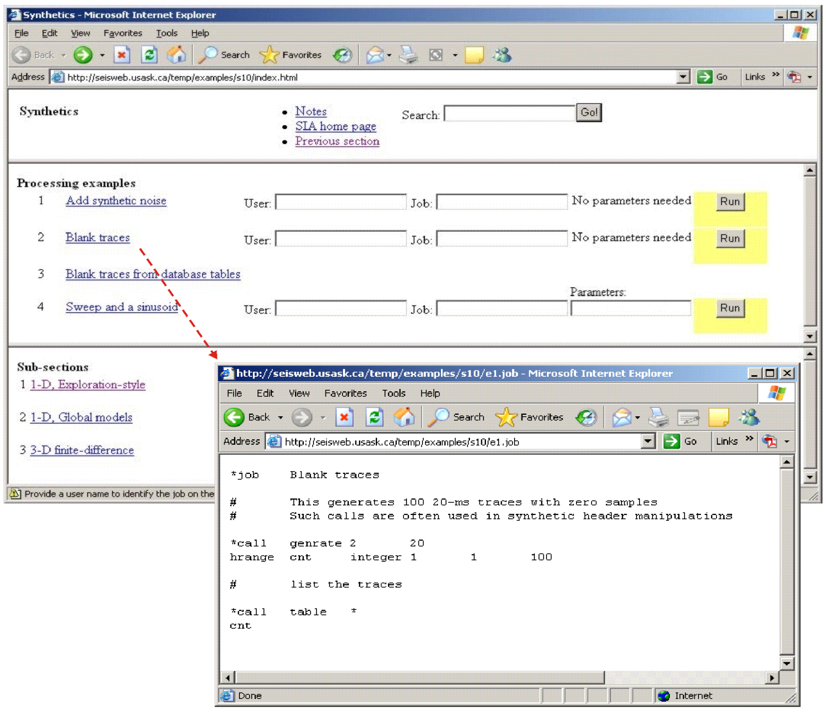
The application **plotmtv** (http://www.phy.ornl.gov/csep/CSEP/CORNELL/TUTORIAL/PLOTMTV/OVERVIEW.html) is a fast multi-purpose plotting program for visualization of scientific data in an X11-window environment, which also produces useful PostScript graphics. We created a seamless interface for this application and use it to view database tables and seismic traces. We also bundle *plotmtv* into the standard SIA distribution.

The **Generic Mapping Tools (GMT)** is a collection of ~60 UNIX tools for high-quality geoscience PostScript graphics, and particularly maps (see example in Figure 6.1). It was incorporated as one of the rendering "drivers" in the SIA graphics system. In addition, the display server is also able to access GMT georeference databases directly and rendering them in full 3D using OpenGL (Figure 4.4).

Rayinvr (Zelt and Smith, 1992) is a popular travel-time modeling and inversion program for wide-angle seismic data. It was incorporated by creating model and travel-time editors, introduction of corrections for crooked-line and ellipsoidal-Earth geometries. Work on interactive ray-tracing in true 3D geometry is underway (Figure 4.4).

6.7 Data processing and modeling web services

SIA is also apparently the first seismic processing system to operate as a web service (Morozov et al., 2006). A standard distribution installed on a system accessible via HTTP (e.g., http://seisweb.usask.ca/SIA/ps.php) can receive processing jobs, execute them and return the results, currently in the form of web pages or files ready for download (Figure 6.7). The content of this processing is entirely controlled by the client. The client is even able to upload web forms on the server and associate them with processing jobs, thereby creating custom web data or processing services. This approach was utilized to generate a library of SIA processing examples, some of which are also executable on-line (Figure 6.7; also see http://seisweb.usask.ca/temp/examples).


Synthetics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://seisweb.usask.ca/temp/examples/s10/index.html Go Links
http://seisweb.usask.ca/temp/examples/s10/e1.job - Microsoft Internet Explorer
Address http://seisweb.usask.ca/temp/examples/s10/e1.job
Done
Internet

Synthetics

- Notes
- SIA home page
- Previous section

Search: [ ] Go!

Processing examples

1	Add synthetic noise	User:	Job:	No parameters needed	Run
2	Blank traces	User:	Job:	No parameters needed	Run
3	Blank traces from database tables				
4	Sweep and a sinusoid	User:	Job:	Parameters:	Run

Sub-sections
1 1-D, Exploration-style
2 1-D, Global models
3 3-D finite-difference

Provide a user name to identify the job on the

```
*job    Blank traces

#       This generates 100 20-ms traces with zero samples
#       Such calls are often used in synthetic header manipulations

*call   genrate 2       20
hrange  cnt     integer 1       1       100

#       list the traces

*call   table   *
cnt
```

Figure 6.7 Sample page (section "Synthetics," larger window) in the current library of processing examples (http://seisweb.usask.ca/temp/examples). Such pages are generated by tool 'expert' included in the processing flows executed on the server. The contents of on of the sample flows are shown in the smaller window in Disco-like format.

6.8 Development framework

From its inception, SIA was not intended as a complete product to serve a specific narrow task, such as reflection seismic processing (Morozov and Smithson, 1997). Instead, the design goal was to provide an extensible framework capable of supporting nearly any type of geophysical data processing, modeling, or interpretation. However, due to the character of its previous applications, most of SIA toolkit development was so far related to seismology.

The system allows its users to rapidly add new functionality with a minimal effort. Two principal features simplify the development within SIA. Firstly, new modules can be added to perform custom data processing while taking advantage of other tools and extensive C++ class libraries, including Qt and OpenGL graphics. Secondly, tool interactions, aided by the GUI, effectively transform Disco-like job scripting into a model- and process-description language. Custom interactive graphical applications can thus be created by simply designing processing flows and without any "serious" computer programming.

New tools can be coded using a mixture of C, C++, FORTRAN, and even Pascal or Java. At the University of Saskatchewan, graduate students routinely write new processing modules for class exercises and also to further their research. In our experience, a reasonably complex tool can be completed in only a few days. Templates have been created to aid in the development process, and a complete set of compilation and linking tools are provided. New modules are integrated with the system by the maintenance utilities so that they become available from the graphical interface and provide fully functional context-sensitive help to the user.

6.9 Automated documentation, code distribution, and collaboration tools

Given its role as a development framework, the central theme of SIA in recent years has actually been code and documentation maintenance. With about 600,000 lines of tightly integrated code, special efforts are required for facilitating development, maintaining user documentation, and performing system integration and testing. Most of these services are wrapped into a single utility `sia-update`, which can be used to compile system libraries, tools, the GUI or display packages, and test them. The utility also creates user's and programmer's documentation (http://seisweb.usask.ca/SIA/index/), posts examples, and creates new user setup. First-time installation or update from a remote distribution can be performed by a single call to `sia-install` utility, which can be obtained from the SIA installation page at http://seisweb.usask.ca/SIA/doc/install.html.

To aid in decentralized collaborative development, SIA offers an automated code distribution system (http://seisweb.usask.ca/SIA/cs.php) modeled after open-source projects such as `apt-get` and `yum`. Each installation may configure a list of repositories which will be checked for updates to currently installed or new tools. If updates are available, the user is notified through the GUI and is provided with their descriptions. When an update is selected for installation, the source code is downloaded from the server and compiled on the local system. The entire process is automated and controlled from within the GUI, or it can be performed from a command line. By downloading source codes rather than binaries, the system is able to share tools across many supported architectures. Further, the code is compiled optimized for the hardware it is running on (i.e. AMD, Intel, or PowerPC) ensuring the best possible performance. The ability to install and update code is restricted to "administrative" users, which may be useful where there is a single installation for a number of users.

Well-supported open-source code standardization could allow multiple developers to collaborate by sharing the codes in a consistent, reliable, and architecture-independent manner. SIA accomplishes this by allowing *any* installation to be used for code development and also to function as a code server (if a standard web server, e.g., Apache, is available). In such a way, source codes developed locally become immediately available for installation on all subscribing systems. Finally, the author of a new tool can arrange for automatic "bug reports" related to that tool to be received by the code web service above.

6.10 Conclusion

SIA appears to be the most full-featured seismic processing system which could be of interest to researchers in both academia and industry. Its strengths are in its unique processing concept, broad scope, modern interface, robust core, very general visualization system, and parallelization capabilities. Since new ideas in seismic processing constantly require new software, SIA is optimized to serve as a concurrent development framework allowing new processing tools to be rapidly developed while leveraging the existing code and graphical utilities to dramatically reduce the time and effort required. The display system seamlessly handles both 2D and 3D data while offering some unique features and allowing extensive customization by the user without the need for programming. A code update and distribution system provides easy and automated access to software updates and allows developers to share their work without the need for installation or maintenance utilities.

As a closing remark, note that unlike the FreeUSP, GMT, SEP, Seismic Un*X, SIOSEIS, and of course their commercial analogs, practically everything of the above was accomplished without any financial support. The development was carried out in support for different projects in several areas of geophysics, united with a firm belief that the software can and shall be well-designed, integrated, re-used, and shared.

7 Rebuilding a Regional Seismographic Network in Southern Saskatchewan

In this and the following Chapters, I touch on a specific application of the software development described above. Chapter 7 is based on the paper by Morozov et al. (2007), in which we present the need for seismic monitoring in Saskatchewan and the current status of rebuilding the network by using new digital instruments. I built the first two instruments myself, and proposed an Internet software solution described in Chapter 8 (based on Chubak and Morozov, 2008). This system is not directly related to IGeoS package; however, I designed an Internet data exchange protocol, and my supervisor (I. Morozov) wrote several IGeoS tools to use it.

Currently, two stations are functional near Saskatoon, with seismic data continuously transmitted to the lab, processed in real-time by IGeoS flows, and saved on disk and regularly backed up. Note that such level of automation appears to be unavailable with commercial hardware and software solutions provided, for example, by Nanometrics and employed by the Canadian Seismograph Network. However, such automation is critical in our environment, in which there are no personnel regularly monitoring the seismic data acquisition.

7.1 Introduction

Despite its large territory, Saskatchewan shows a striking paucity of seismic monitoring stations compared to practically any other area in North America. Saskatchewan is the only province in Canada not contributing seismic data to the Canadian National Seismograph Network (Figure 7.1). Although mining and petroleum exploration companies routinely conduct local micro-earthquake monitoring for hazard mitigation and assessment studies, no regional seismic data is systematically collected for research purposes. In the 1960's-70's, the Department of National Defence operated a station near Creighton for atomic blast monitoring. Since 1978, regional stations were operated by the University of Saskatchewan at four different sites on the surface and two underground (Agrium and Colonsay mines). The Geological Survey of Canada operated the Big Muddy station from about 1982 to 1990. Only one of these stations remains functional at present.

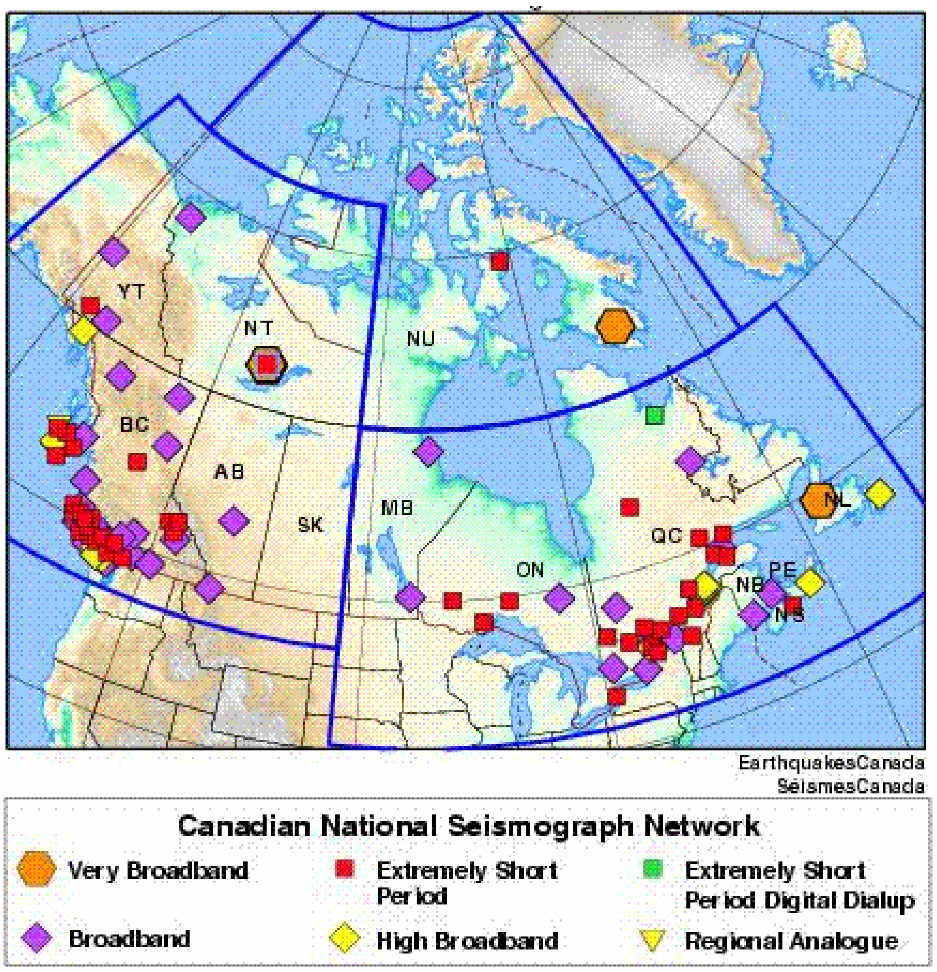


Figure 7.1 Map of the Canadian National Seismograph Network (from Natural resources Canada web site, http://earthquakescanada.nrcan.gc.ca). Note the gap in station coverage across Saskatchewan.

The paucity of seismic recording in Saskatchewan is explained by several reasons. First, together with Manitoba, the province is among the least seismically active in North America (Figure 7.1). The largest known earthquake in Saskatchewan was the M = 5 1/2 event in 1909. Since the start of instrumental recording in western Canada in the mid 1960's there have been 13 known natural earthquakes. Since 1978 there have been more than 40 mining induced earthquakes, some causing minor damage. Also, the strength of the seismology groups at the University of Saskatchewan has traditionally been in active-source, exploration, and engineering seismics. Maintaining several continuously monitoring seismic stations requires a well-developed infrastructure and constant attention by skilled personnel. Both of these requirements are contingent on funding which had only been intermittent for earthquake-related projects.

Earthquakes of magnitudes over about 2.5 are not uncommon in Saskatchewan (Figure

7.2). In recent months, several such earthquakes were recorded by the existing U of S station and local networks operated by potash mining companies. To accurately locate seismic events, invert for their source parameters, and interpret their nature, one needs wider-aperture and lower-frequency seismic recording than the one used for mitigation of mining hazards. In addition, analysis of such data needs to utilize the observations from other national and international networks. At present, GSC location accuracy in southern Saskatchewan is ±20 km, and the analysis of small events is poor.

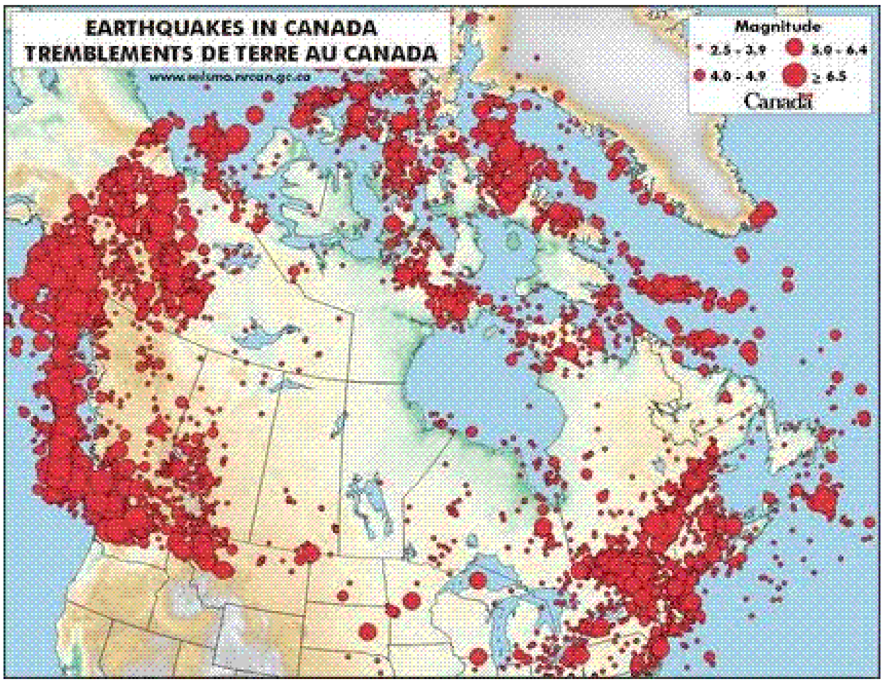


Figure 7.2 Map of Canadian earthquakes (from Natural Resources Canada web site, http://earthquakescanada.nrcan.gc.ca).

Apart from monitoring the local seismicity, a permanent, modern, and robust seismic network in Saskatchewan would help filling the gap in seismic data coverage across Canada (Figure 7.1). The existing U of S station responds to earthquakes of magnitudes ~4-5 from the continental margins of Canada and from the Arctic (Figure 7.2), and events of over m≈5.5 – 6 are typically well-recorded worldwide. Events as small as m = 2.5 can be recorded anywhere in Saskatchewan

Strong seismic events occurring at large distances (20-50^o) can be used not only to analyse the deep interior of the Earth but also to provide valuable information about the structure of the crust and even the crystalline basement. Modern data analysis techniques allow inversion for near-station structures that might be of interest for diamond exploration and for petroleum industry. Thus, by using the so-called Receiver Function technique, one can combine recordings of different components of ground motion to measure the thickness of the crust and of basin sediments. Crustal anisotropy measured from teleseismic recordings provides information about tectonic stress. Also, as it has been shown recently, cross-correlation of the seismic “noise” (Bensen et al 2008) from several stations can be used to invert for crustal (in our case, basin) structures.

Another important aspect of expanding seismic monitoring activities is education and outreach. Real-time seismic recording can be naturally combined with live and interactive displays showing seismic data, recent earthquakes, and also providing various geology-, science-, and resource-related information. Such displays are becoming increasingly popular and are getting recognized as invaluable educational tools. We have recently set up such a display near the Museum of Natural Sciences at the U of S (Figure 7.3).

For all aspects of seismic monitoring mentioned above, from real-time monitoring to advanced analysis and computer displays, fully digital recording and automated data handling is required. The existing U of S station near Bergheim (Figure 7.4) is analogue and uses a helicorder (paper drum recorder) to record seismic waves. Although simple and robust, this system does not allow reproduction of the records in a form suitable for further analysis or remote display. Recent upgrading of this station to a digital Taurus seismograph by Nanometrics did not completely resolve this problem, as the system still requires extensive and continuous effort for saving and displaying the records.

Here, we describe our approach to rebuilding the U of S seismographic network, converting it into modern digital technology, and integration with live public-interest and educational displays. We decided to pursue this goal by building our own low-cost instruments and by developing all the necessary software for data acquisition, processing, and display. In conclusion, we outline our plans for its further expansion and enhancement.

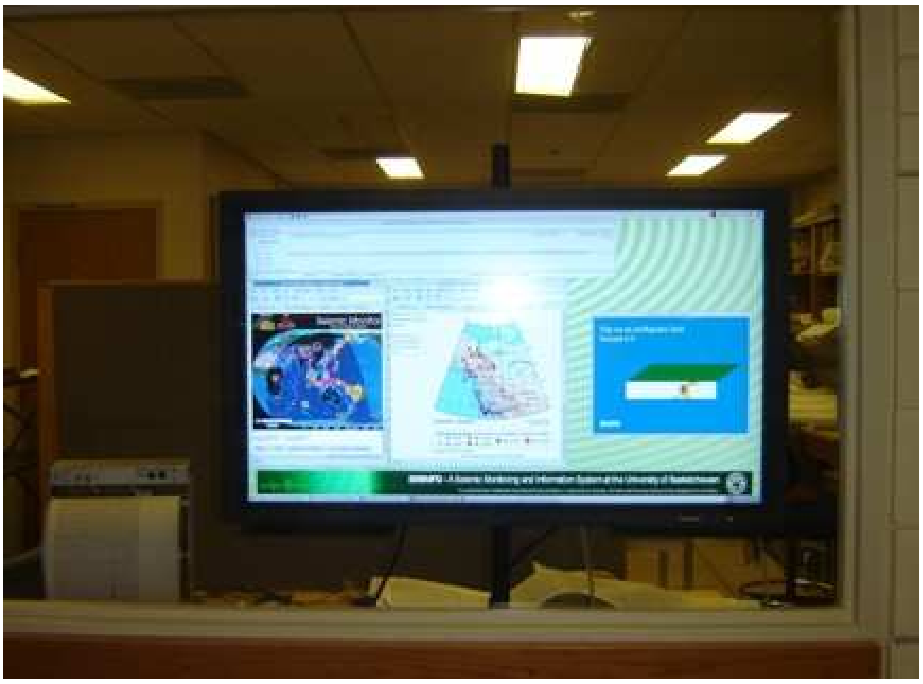


Figure 7.3 Display of earthquake-related information at the Department of Geological Sciences, University of Saskatchewan. This live web-based TV display constantly shows the recent

global earthquakes, live seismograms from our SKBG station (Figure 7.5), as well as presentations about the Earth, tectonics, and seismology.

7.2 Low-cost Internet seismograph

To achieve an affordable and low-maintenance solution for seismic data acquisition, we built our own system using components (PAR4CH 4-channel amplifier, 24-bit A/D converter, and a GPS clock) manufactured by Symmetric Research (http://symres.com). The use of this hardware allowed reducing the cost of the system by ~8 times compared to a Taurus while providing similar, industry-standard data quality. The seismograph boards were connected to a mini-ITX PC computer. No moving parts (fans or hard disk drives) were used in order to reduce vibration that might influence the recordings. The equipment was mounted inside the upper half of a steel barrel, with its bottom part occupied by three 1-Hz geophones by Geospace (Figure 7.4). For temperature control during the cold season, two long-life incandescent light bulbs are used (Figure 7.4).

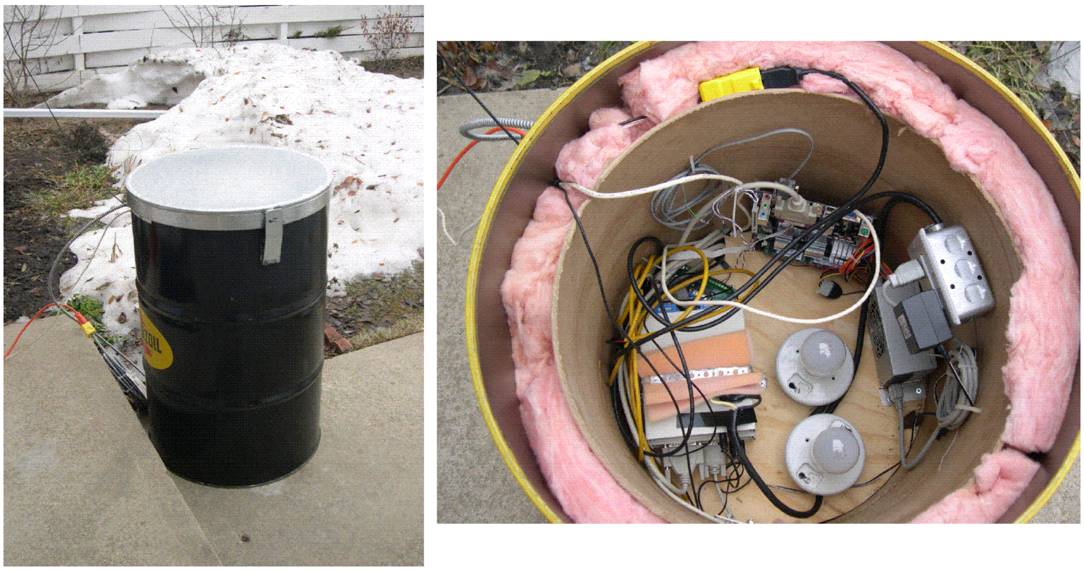


Figure 7.4 Left: the seismic station during testing. Right: the seismograph assembly, with its lid open. Thermal insulation and electrical bulbs are used for temperature control during Saskatchewan winters.

Open-source Linux operating system was installed on the computer, with device drivers provided by Symmetric Research. We also wrote all the necessary software for data acquisition and transmission to the data collection facility via a standard Internet (wired or wireless) connection. The software consists of five main parts: 1) data collection server installed on the acquisition computer (Figure 7.4); 2) web server allowing remote control of the acquisition software; 3) data relay program installed on a Linux computer in the lab, 4) ring buffer for continuous disk data storage, and 5) data analysis and visualization software.

The data acquisition server program constantly monitors the network for an available data recipient (typically, the relay program) and sends the seismic records to it in near-real time. In addition to the records of ground movement, GPS timing data (to sub-1 ms accuracy) and state-of-health information (such as the temperature inside the system compartment) is transmitted. If the network is down, the server stores the records and attempts resending them when the connection is re-established. An Apache web server installed on the same computer allows viewing and setting parameters of the recorder remotely, by using any web browser. Finally, whenever a hazardous system condition develops (e.g., temperature dropping out of range), the system sends an email to alert the administrator and if needed, performs data backup and prepares for shutdown.

The data relay program receives a continuous stream of data from the field unit and re-broadcasts it to one or several clients interested in data analysis or display. The program also saves the records continuously to disk in the form of a "ring buffer" allowing random or circular access to the data by other programs. A single relay program can serve any number of field data servers, and in the future, it will also be able to send the data to the national data centres and to other interested parties. State-of-health information is also saved in a database immediately as it is received.

The data analysis software is very flexible, broad in scope, and takes advantage of the power of SIA seismic processing systems that we have developed over a number of years (e.g., Morozov and Smithson, 1997; Chubak and Morozov, 2006; http://seisweb.usask.ca/SIA). The system currently includes about 200 tools for data filtering, inversion, and display, and it was used in a number of applications ranging from exploration, crustal and earthquake seismology to the analysis of gravity and air-magnetic data (Li et al., 2005), 3D data visualization, and even web services (Morozov et al., 2006). By adding a module for network data input, we obtained a variety of ways for displaying or saving the records. The specific choice for data display is made by the user by designing the appropriate combination of SIA tools (Figure 3.3). In addition, the software is being continuously expanded by the Geophysics graduate students as a part of their class and research projects.

7.3 Towards a digital seismic network near Saskatoon

The existing and proposed sites for the new U of S digital seismograph stations are shown in Figure 7.5. SKBG is the station located in the U of S Geophysics test site near Bergheim. The station is equipped with a 3-component Nanometrics Taurus seismograph, the same as used by the national POLARIS consortium in Canada (http://www.polarisnet.ca/). Currently, the station is using three 1-Hz L-4 geophones, which we intend to upgrade to a broad-band sensor in the future. The data from Taurus are streamed to the U of S via a radio Internet link. An additional vertical-component analogue channel still also operates at this station.

Station SKWC is being installed at the time of this writing (April 2007) at the White Cap Dakota First Nation grounds south of Saskatoon. In addition to recording regional and global seismicity, the purpose of this station is to provide additional information for monitoring ammunition blasting activities at the Canadian Forces depot near Dundurn.

Power connection for this station was provided by the Forces, which are also committed to partial maintaining of the power and 24-hour wireless Internet service to it. Sites for the third station are sought near Colonsay (Figure 7.5); this location would be optimal for triangulation required for accurate location of seismic events.

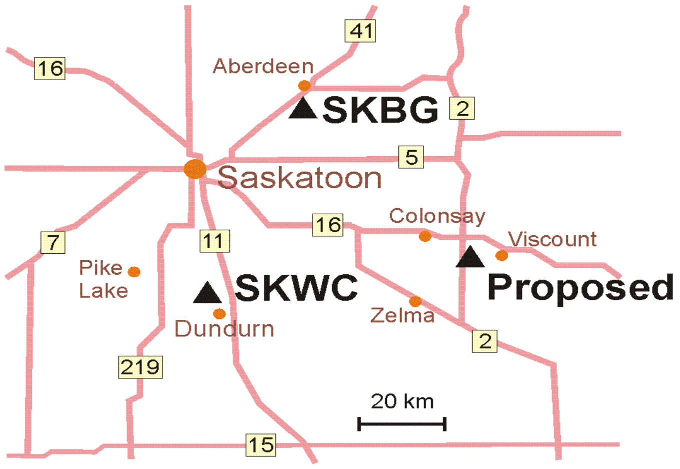


Figure 7.5 Existing and proposed digital seismic stations near Saskatoon.

For consistent operation of a seismic network by a small University program, a high degree of automation is required. Our goal is to achieve continuous and generally automatic data acquisition, archiving, event detection, generation of event bulletins, extraction of event windows, and displays. To achieve this, we have programmed an STA/LTA (Short-Time-Average/Long-Time-Average) event detection algorithm and also methods for record extraction and display. The resulting continuous seismic records as well as time windows of extracted events can be shown on the public seismic display (Figure 6.5).

With small cost, high degree of automation, expandability and flexibility, the approach appears to be ideal for further expansion of the regional seismic network in southern Saskatchewan. Additional stations would improve the ability to detect and accurately characterize seismic event, and contribute important data for student training and public interest and education. With the design described above, only quiet locations with AC power and Internet connections are needed, and neither of these requirements is difficult to satisfy in Saskatchewan. Currently, we are looking for three types of land owners who may, in our opinion, be interested in and benefit from installation of such seismic stations: 1) mining (particularly potash) operations; 2) high-speed Internet provider networks; and 3) rural high schools. The last of these options is particularly attractive, as it would provide the students with a unique opportunity for hands-on research related to their land, to participate in an exciting, quantitative natural science, and at the same time to make tangible contribution to the global activity for monitoring earthquakes.

8 Low-cost continuous seismic acquisition utilizing open-source software

In this Chapter, in I describe the design of the seismograph that I built recently, with particular emphasis of the real-time Internet data exchange. The Chapter is based on the paper by Chubak and Morozov (2008). The key points of the approach are its low-cost (the complete system costs about 6 times less than comparable Taurus seismograph by Nanometrics), open-source software, high degree of automation, and seamless data streaming in a customizable data processing and analysis. As a sample of such analysis, I present my implementation of the so-called STA/LTA event detection algorithm.

8.1 Introduction

Continuous monitoring of regional seismicity is important for locating earthquakes and the mitigation of earthquake hazards. In Saskatchewan, this is currently only performed by our group at the University of Saskatchewan. We aim at a robust, low-maintenance and low-cost solution with full automation of data acquisition, archiving, and processing. This is achieved through building inexpensive multichannel digital data loggers and utilizing open-source software to transmit the continuous records over a TCP/IP network connection.

The system consists of three components: acquisition data server (located near the geophones), data relay program (located in the data centre) and client programs used for displaying, processing, and saving the data. Various clients are available, from simple display tools to a direct feed into the IGeoS processing package. This allows practically unlimited flexibility of processing applied to the real-time data stream, from immediate archiving to creating ring buffers, identifying events, or producing various data displays. Web server is used to display system status and set acquisition parameters. Currently, the system has two operational stations near Saskatoon, Saskatchewan, with more stations planned.

As a seismic monitoring system operated by a small University group with, our hardware/software solution is designed to be a reliable and fully automated hardware and software combination for collecting, transmitting, processing and storing seismic data. While use of the system has been focused on a serving as a regional (1Hz) seismograph, the network design and concept of direct feed into processing should also work well in reservoir monitoring. The modular approach (server, relay and client) and documented interface allow new clients to be written to integrate real time data in other applications which could be useful even outside of the seismic community.

Considerable effort was spent to ensure that the system uses a minimum of network bandwidth which makes it suitable for slower internet links found in remote locations. To keep operating and maintenance costs at a minimum, the data is processed without the need for an operator and events are automatically identified. Most significantly, the data can be loaded directly into a full-featured geophysical processing package (IGeoS). This allows a great deal of flexibility in the processing scheme. The software is open-source (Linux-based), and the hardware is build from standard computer components and an inexpensive commercially available 24-bit A/D system with GPS timing.

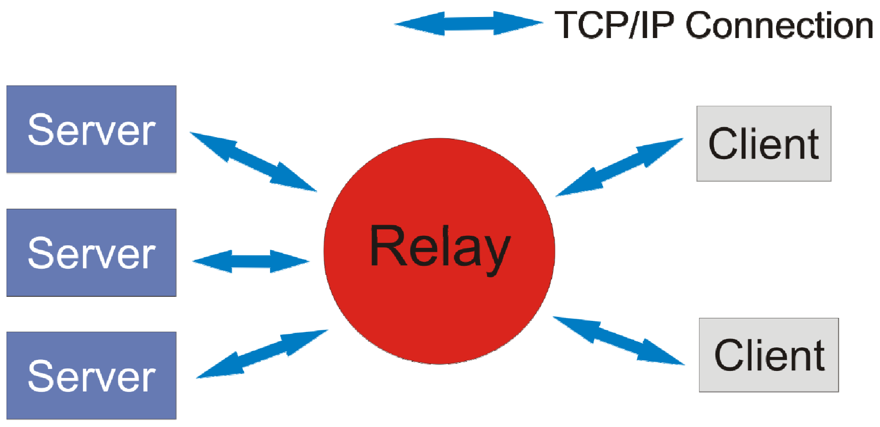


Figure 8.1 Seismic data network design

Three distinct components comprise the software: server, relay, and client (Figure 8.1). Communication is handled over a standard TCP/IP network connection. The server component is located at site of the data acquisition and transmits the digitized data to the relay. Multiple servers can connect to a single relay which can pass the data unaltered to the clients or first perform some timing synchronization. Clients can connect to the relay and retrieve a list of available servers. They are then able to specify which servers to receive the data stream from.

8.2 Server

The data acquisition server program is automatically started on boot, makes a network socket connection to the relay program and begins to communicate the digitized values, time marks and GPS coordinate and time strings as they are retrieved from the hardware. Timing data may arrive significantly before or after the samples which they describe, and thus it is necessary to synchronize the data before it is used. However, in order to keep the server as simple and robust as possible, the synchronization is done off site, by the relay or client programs.

If the network connection is broken, the data can be stored locally for later transmission or retrieval via a web browser. When the network becomes available again, the connection with the relay is reestablished automatically, and the transmission continues. Optionally, the server can back up all data to a local disk in addition to transmitting it.

To monitor the state of health (currently temperature) of the system, we use the 1wire devices from Dallas Semiconductor. A model DS9097 serial to 1wire adapter is used and a DS18S20 digital 1wire thermometer. The server software polls the thermometer at a configurable interval and transmits a "state of health" package to the relay which includes the time and temperature. If the temperature is outside of the configured high or low alarm points, an email is also generated and sent to the specified addresses.

8.3 Relay

The data relay program runs continuously at the data center computer and accepts network connections from all data servers and from clients in the system (Figure 8.1). It has two main functions: re-distributing data and synchronizing the timing. It acts as a distribution point for the data which reduces the load on the internet connections from the servers. With this design, only a single stream needs to be sent from the remote site regardless of how many clients are receiving the data. Communication is formatted using a small set of XML tags.

Each server makes a connection to the relay, identifies itself as a data source and provides site information. The available site names are then transmitted to any clients that connect. Clients connect to the relay and specify what type of data (e.g., raw or time-synchronized) they expect. Synchronization is accomplished by creating a data queue for both the time marks and the corresponding data packets in the relay.

8.4 Clients

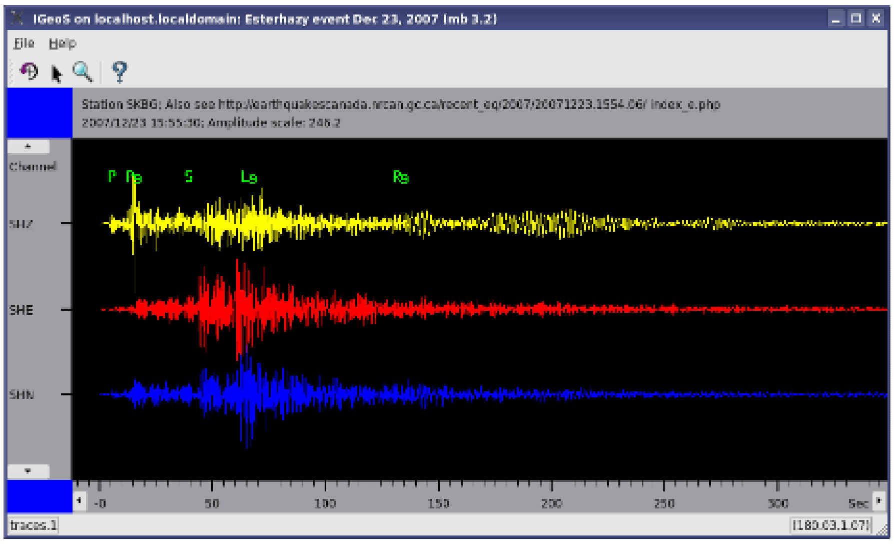
Implementation of a client for this system is relatively simple: the program must only be able to make a socket connection over a network to the relay and send and parse a few XML tags. We have written a simple client which displays the real-time data from selected channels. Another client program send the real-time data into the IGeoS processing package (formerly SIA, Morozov and Smithson, 1997; Chubak and Morozov, 2006), which allows to perform any standard seismic processing and leverage the more than 200 tools currently in the package. Because the data is fed directly into a processing package, the result is limited only by the selection of tools made by the user. For example, we use processing flows which save the data to a RAID concurrently with performing filtering and preprocessing and applying an STA/LTA (Short Term Average / Long Term Average) event detection. IGeoS client processing flows can also use 3D OpenGL visualization or PostScript to display the data, and to produce various types of file outputs, such as formatted in ASCII, SAC, or SEG-Y.

8.5 STA/LTA Event Detection

Seismic events are auto detected by a SIA tool utilizing an STA/LTA algorithm. First, “short” and “long” window lengths are defined depending on the wavelength of interest. Testing has shown that using a short window of approximately ¼ of the expected wavelength and a long window of 100 times larger produces useful results though a great deal of precision is not required. Depending on the noise in the system it may be useful to have the short window cover several complete wavelengths. In this way short noise is less likely to trigger an event. The signal is smoothed to reduce false triggers from noise and the average absolute value of the signal is calculated for both the short and long windows to produce the short and long term averages. During an event the Short Term Average (STA) increases much faster than the Long Term Average (LTA) so the ratio STA/LTA can be used to indicate an event. When the value of STA/LTA passes a defined threshold an event is declared.

8.6 Current Installation

Mining and other human activities account for many of the seismic events in Saskatchewan. These are of interest to the public and in many cases to the exploration community as well. The first system we installed is located on the Whitecap Reserve south of Saskatoon, SK in response to concerns that seismic activity from a nearby military base might be affecting the structural integrity of the buildings (Figure 7.5; Morozov et al., 2007). A second station is located at the UofS Geophysics test site east of Saskatoon. From this location we have recently recorded a large, m_b=3.2, seismic event near Esterhazy, SK (Figure 8.2). We are currently looking for a site for the third station of the network, which is necessary for accurate event location (Figure 7.5).



IGeoS on localhost.localdomain: Esterhazy event Dec 23, 2007 (mb 3.2)

Station SKBG: Also see http://earthquakescanada.nrcan.gc.ca/recent_eq/2007/20071223.1554.06/ index_e.php
2007/12/23 15:55:30; Amplitude scale: 246.2

Figure 8.2 Esterhazy, SK event on Dec 23, 2007 recorded at station SKBG in Figure 7.5

8.7 Conclusion

While network monitoring systems are not new, we believe that the use of open-source software combined with commodity hardware provides low-cost and robust solution for remote seismic acquisition. The unique (to our knowledge) integration of the data stream into a processing package provides features and flexibility not found in other systems. Finally, as a complete solution for seismic monitoring it is necessary only to provide the appropriate hardware and a location with power and internet to begin collecting and analyzing data.

9 Discussion and Conclusions

As a result of the contributions from this Thesis, development of IGeoS package is sufficiently complete for it to be useful to geophysicists working on a variety of tasks. However, IGeoS's contribution to the community may not be limited to its current functionality as it provides a framework to develop, manage and maintain geophysical software. By developing code within a framework, the need to duplicate existing work is avoided and the installation and maintenance of code is simplified.

The graphical interface significantly reduces the amount of time a new user needs to spend to learn the scripting language. Since help functionality and documentation are provided through the interface, it is possible to build and modify flows without the need for extensive training.

3D visualization is provided through an abstracted OpenGL implementation which allows IGeoS to serve not only as a processing system but as an interpretation and display application. Semi-custom applications can be created by using the processing flow to link widgets such as buttons or sliders to the inputs of other tools. This allows researchers to rapidly produce an application which will perform tasks not found in other packages without the need for any programming.

To effectively acquire and transfer seismic data from remote locations, a collection of network tools were written. Their modular nature allows the tools to function independently or within the IGeoS framework. When used with IGeoS, the data stream is incorporated directly into the seismic processing flow so that standard processing steps can be applied to the data. Multiple streams can be analyzed within a single processing flow to locate events.

Finally, as most extensive development efforts, this project paved the way for much more potential further developments. Although IGeoS appears to be conceptually complete with all major components and functionally designed, significant work is still required to maintain and enhance many of the features within the code. Additional tools for PVM, multi-processor, and multi-core parallelization need to be developed. Many of the existing tools could benefit from the new interactive and graphical functionalities, and many new data analysis approaches may emerge as a result of the enhanced capabilities of the package. As an example, the development of a new 3D refraction statics package (Jhajhria, 2009) has been largely facilitated by the ability to analyse the first-arrival travel-time surfaces interactively in 3D. In the real-time data acquisition project and exciting development would be to implement continuous data delivery to the Canadian Seismograph Network, expanding the system, and making automated location and parameter estimation of earthquakes.

9.1 Suggestions for further research

While IGeoS is complete in terms of providing facilities for the organization, processing, visualization and interpretation of geophysical data there are numerous areas which I would like to see enhanced. In a commercial environment reproducibility is of little concern but to researchers who need to be able to reproduce results it becomes paramount. To address the concern of these users, IGeoS would need to take one of two approaches: First it could provide a strict version control system on all code submissions with a means of specifying a particular version for use in processing. In this way it would be possible to use the same version of code later to reproduce an experiment. This is the more rigorous approach but is rather cumbersome in practice. If all code was stored in CVS this would be readily possible but rather awkward to use. A test driven development model is the Second approach and the one used the Madagascar software package. All code is written with a test suite and any revisions of the code must still produce the same output from the test suite. While this is a workable approach it still does not guarantee that there are not errors which may affect results but not show up on the test suite. Further it becomes difficult to correct errors or enhance a tool as the data output may vary.

Packaging IGeoS in some of the standard systems such as RPM (Redhat Package Manager) or .deb (Debian Packkage Manager) would greatly improve its availability to end users. This requires some effort as the package needs to be continuously updated as the linux distributions change. Another possibility is to work towards making IGeoS available for the windows platform. The QT components should work fine but it would currently be necessary to use an environment such as Cygwin to provide Unix functionality for certain functions.

10 References

Bohlen, T., 2002; Parallel 3-D viscoelastic finite difference seismic modeling, Computers & Geosciences, 28, 887-899.

Bensen, G.D., M.H. Ritzwoller, and N.M. Shapiro, Broad-band ambient noise surface wave tomography across the United Stated, J. Geophysical Research, 113

Chubak, G., and I. Morozov (2006a) Integrated Open-Source Geophysical Processing and Visualization, 2006 CSEG Convention, Calgary, AB, May 2006.

Chubak, G., and I. Morozov (2006b). Integrated software framework for processing of geophysical data, Computers & Geosciences, 32, 767-775.

Chubak, G., and I. Morozov (2007). Automated maintenance of geophysical software from distributed web repositories, Computers & Geosciences, 33, 835-37.

Chubak, G., and I. Morozov (2008) Low-Cost Continuous Seismic Acquisition Solution Utilizing Open-Source Software, 2008 CSEG Convention, Calgary, AB, May 2008.

Chubak, G., I. Morozov, and S. Blyth (2007a) Integrated Open-Source Geophysical Processing and Visualization, 2007 CSEG Convention, Calgary, AB, May 2007.

Chubak, G., I. Morozov, and S. Blyth (2007b), Towards a comprehensive open-source system for geophysical data processing and interpretation, CSEG Recorder, 32, Sept 2007, 52-62.

Fuchs, K., & G. Müller (1971), Computation of synthetic seismograms with the reflectivity method and comparison with observations, Geophysical Journal of the Royal Astronomical Society, 23, 417-433.

Kennett, B. L. N., 1993; Seismic Waves Propagation in Stratified Media, Cambridge University press.

Morozov, I. B. (1998), 3D seismic processing monitor, Computers & Geosciences, 24, 285-288.

Morozov, I. B., & S. B. Smithson, (1997), A new system for multicomponent seismic processing, Computers & Geosciences, 23, 689-696.

Morozov, I. B., Smithson, S. B., Chen, J., and Hollister, L. S. Generation of new continental crust and terrane accretion in Southeastern Alaska and Western British Columbia from P- and S-wave wide-angle Seismic Data (ACCRETE), (2001). Tectonophysics, 341/1-4, 49-67.

Morozov, I., G. Chubak, & S. Blyth, (in press) Interactive 3D/2D visualization for geophysical data processing and interpretation, Computers and Geosciences, http://dx.doi.org/10.1016/j.cageo.2008.10.005, assessed Feb 15, 2009

Morozov, I., B. Reilkoff, and G, Chubak (2006). A generalized web service model for geophysical data processing and modeling, Computers & Geosciences. 32, 1403-1410.

Morozov, I., G. Chubak, and L. Litwin, 2007. Rebuilding a regional seismographic network in southern Saskatchewan; in Summary of Investigations 2007, Volume 1, Saskatchewan Geological Survey, Sask. Industry Resources, Misc. Rep. 2007-4.1, CD-ROM, Paper A-1, 8p.

Morozov, I.B., & K. G. Dueker (2003), Depth-domain processing of teleseismic receiver functions and generalized three-dimensional imaging, Bulletin of the Seismological Society of America, 93, 1984-1993.

Stockwell, Jr. J. W. (1999), The CWP/SU: Seismic Un*x Package, Computers & Geosciences, 25, 415-419.

Templeton, M. E. & C.A. Gough (1999), Web Seismic Un*x: Making seismic reflection processing more accessible, Computers & Geosciences, 25, 285-288.

Wessel P., & W. H. F. Smith (1995), New version of the Generic Mapping Tools released, EOS Trans. American Geophysical Union., 76, 329.

Zelt C.A. & R.B. Smith (1992), Seismic travel-time inversion for 2-D crustal velocity structure, Geophysical Journal International, 108, 16-34.

11 Appendices

11.1 Appendix A. Job example: 2D ray tracing model and GMT base map in 3D (Figure 4.4)

As an example, we present fragments of the SIA job used to produce the 3D display shown in Figure 4.4 We use the traditional DISCO-style job format, in which parameters are identified by the positions of the corresponding tab-delimited input fields. Tool descriptions start at tokens '*call'. Note that tool parameterizations can be extensive and are structured by providing multiple "parameter lists" identified by the corresponding keywords. Some of these lists are indicated by commentaries in the example. Lists themselves may also span multiple lines and contain complex parameterization of the display.

```
#############################################################
#        ACCREETE model in 3D
#############################################################
###### Job set-up. In particular, use kilometers for distance units
###################################################################
*setup
noapp
units km    km

###### load the profile track line and midpoints from ASCII files
#################################################################
*call readtab table   x-profile
x     real  1
lat   float
lon   float
file    all     pc-coords.txt
*call readtab table   midpoints
lon   float
lat   float 1
z     float
file    all     ../MAP/midpoints-rev.table
      z     0.    # draw midpoints at z=0.0

### load the interpolated Moho depth grid
#####################################################
*call readtab     map   moho
lon   float 1
```

```
lat   float 2
z     float
argrang     lon   -131.3      0.01  111
argrang     lat   54.6  0.01  61
file  all   moho_depth.xyz

###### create graphic elements
#####################################################

*call graphic
backgr      black
foregr      black

# line and fill styles:
line  solid 1     gray        line-shore
line  solid 1     white       line-borders
line  solid 1     blue        line-rivers
line  dash  1     melon       line-grid
fill  none  blue        fill-wet
fill  none  blue        fill-lakes

# palette for Moho depth:

backgr      -same-
foregr      -same-            # foreground of the same color as top of
the palette

#palette    pal-moho    gmtrainbow        25.0  31.0
#palette    pal-moho    gmtjet                  25.0  31.0
palette     pal-moho    buor              25.0  31.0

# buttons for interactive operation:
button      button_trace      Trace rays
pc-section  exec  trace
button      button_save Save model
pc-section  exec  save
button      button_summary    Print summary
pc-section  exec  print.summary
button      button_display    Display data
pc-section  print This option only prints in this demo!

###### Define 3D projectors
#####################################################
```

```
### this will project 2D images onto profile cross-section:
*call view3d      fence3D
fence x     lon   lat   x-profile
geom  sphere                        km

### this will project 2D images onto the surface:
*call view3d      surface3D
geom  sphere                        km

##### Create surface image
####################################################
*call image surface-maps
range-x     -131  -130        # longitudes
range-y     55    56          # latitudes

# map of the Moho
line  none
surface     lon   lat   z                 moho
pal-moho          z           # fill colors based on z (depth)

# scatter plot of midpoints:
line  points      3     red         line-midpoints
3dtabs      lon   lat   z
midpoints

###### Create 2D Accrete Portland Canal section
####################################################
*incl pc-image.inc

###### Assemble the 3D image
####################################################
*call image acc-3Dimage
rangec3     400               # 200-km viewing box

# list selectable views, the first is the default:
view  From above        -131.5      55    0.    -20   10.   1.0
view  Globe             0     0

# place 2-D cross-section in "fence" projection:
object      pc-section              fence3D

# place 2-D sirface maps in "surface" projection:
object      surface-maps                  surface3D
```

```
# draw in a GMT basemap at "high" resolution:
coast -134  -129  52    58    high
surf  land              fill-dry
surf  sea               fill-wet
surf  lakes             fill-lakes
bndry shore             line-shore
bndry natnl             line-borders
bndry state             line-borders
bndry marine                  line-borders
rivers      r                 line-rivers
grid  line              line-grid   1.0   1.0

# Moho depth color bar:

cscale      vert  0.1   0.8   0.05  0.2   Moho depth (km)
pal-moho

# image lighting:

light 0     0     1000  0.9   Illumination
diff        white
amb         white
spec        cyan

# buttons for interactive operation:

object      button_trace
object      button_save
object      button_summary
object      button_display

# animation (autoplay):

a-rot 0.1   60    30    1.0

##### start the display server
####################################################

*call gui
acc-3Dimage

# request docking windows:
```

```
dockers
otree       canvas            # object tree
otree       graph       # graphics DB tree
prop                    # property editor
```

Note that the job above uses an included file `pc-image.inc` containing a description of the 2D velocity cross-section along the Portland Canal line. This 2D image is projected onto a “fence diagram’ along the line of cross-section and combined with the 2D distribution of a source-receiver midpoints and a coastline plot from GMT. Various editable items, such as coordinates, palettes, colours, and selections of displayed items do not need to be described in the job and are available automatically from the corresponding image objects. Finally, animation (rotation) about a tilted axis in 3D is also illustrated.

Script `pc-image.inc` used in this example, and which can also be utilized for a purely 2D cross-section display, is shown below.

```
### graphics elements

*call graphic
backgr      black
foregr      black

palette     draw-vp1
-0.3  red         0     blue
0     white       0.3   red

# palettes for velocities

palette     draw-vp           gmtseis                5.5   8.4
palhue      draw-vs           color cont  2     5.    4     0.05  .95

line  solid 1     green       layers
line  solid 1     blue        cells-vert
line  solid 1     yellow            cells-diag
line  solid 1     red         rays

fill  solid green       velocity

### RAYINVR 2D ray-tracing tool, loading model from file accrete.v.in1
```

```
*call rayinvr     edit  ps    Accrete
vin   vp                accrete.v.in1
summary

### Form the 2D velocity/interface cross-section image

*call image pc-section        ##### plot the model and rays
range-x     -150  200
range-y     -50   100
#flip-y

axis-x      100         500   annot       border-pen
X (km)

axis-y      10          100   annot       border-pen
Depth (km)

velmod      Accrete
layers                                    layers
cells             vert              cells-vert
cells             diag              cells-diag
vs                                  draw-vs
vp                                  draw-vp

cscale      vert  0.8   0.8   0.05  0.2   Vs    # colour bar
draw-vs

cscale      vert  0.6   0.8   0.05  0.2   Vp    # colour bar
draw-vp
```

11.2 Appendix B. Interactive multi-threaded seismic processing (Figure 4.6)

The following example illustrates seismic trace processing including a simple user interaction. When a button is pressed, one record is loaded and displayed in three different forms: as a time series, spectrogram, and amplitude spectrum (Figure 2.6 that the three transformations are implemented using combinations of various filtering SIA tools, and thus they can be easily customized by the user. Also note that the same tool “plotrt” is used to generate all three displays.

```
#######################################################################
### Simple trace plot example used in real-time viewer development.
### Seismic traces are read in from a file, their headers listed,
### synthetic noise added,
### and the wiggles, spectrogram, and spectra displayed in 2D
#######################################################################

###### begin with optional text definitions

*define
model majin2o     # model name
noise 0
compn 1           # component number to plot
outfile     refl-{model}-0.0001-1.01.sia

###### the following is printed on the screen when the job is started

*info
      Real-time plotting demo for {model}

###### graphics settings and custom elements

*call graphic

# buttons talking to tool “dskrd” below

button      button1           Next trace
dskrd.1           exec  next 1

button      button2           Restart input
dskrd.1           reset
```

```
# custom colors, lines, fills

color 0.2   0.4   0.6   trace-color       # custom color
line  solid 1     yellow            wiggle-line
line  solid 1     red         spectrum-line
fill  solid red         trace-pos-fill
fill  solid blue        trace-neg-fill

# palette ("hot," ported from GMT)

backgr      -same-
foregr      -same-

palette     pal-ampl    gmthot                  -1    1

###### load the records, askthe input to "hold" (wait for button
pressed)

*call dskrd outfile
hold

###### pre-process the records (cut at 1000 msec, count, time-shift,
###### extract one component, apply AGC etc.)

*call modify            1000

*call count chan

# Note the evaluation of trace header expressions:

*call hdrmath
eval  real shift=1000*(chan-1)

*call static      shift apply

*call convert
extract     1

seqind      comp  integer     1     1

*call agc   300

*call noisegn     gauss noise       8     20    80    120
```

```
### Create a data entry point named 'traces_ready'
### for spectral analysis and other steps

*call tee   traces_ready

### recut the traces into 100-ms segments to fit into multichannel
display

*call cuttrc      1000

*call table *
chan  x     shift timstrt     _tsint_     _tnsam_     lasttr

### form the trace display sequences

*call plotrt      traces                 right
channel           5     1000  0.05  0.9   0.75  -0.08 peak  0.5

axis-t      bl    100   500   %1.0f       1

*end  keep        # remove traces, keep header formatting

##################################################################
##### Begin spectral analysis
##################################################################

### Start by reading from 'traces_ready' point

*call tap   copy  traces_ready

### replace the traces with their log(Amplitude Spectra)

*call trcmath
spectr
lg

### measure the peak amplitude and shift the amplitudes down by it

*call trcmath
ampl                    90    100   mean  ampl

*call trcmath
subtr                   ampl
```

```
### Form spectra display

*call plotrt      spectra                 right
combine     replace
channel           1     100.0 0.05  0.07  0.7   0.15  peak  0.8
axis-t      bl    10    20    %1.0f       1

*end  keep  # remove spectral traces

##################################################################
##### Begin spectrograms analysis
##################################################################

### Read again from 'traces_ready' point

*call tap   copy  traces_ready

### Form Rihaczek's instantaneous spectrum

*call spctrgr     rih         50
frange      10    2     100

### Form spectrogram display.
### Note that the spectragram traces are multi-component

*call plotrt      spcgram                 right
channel           1     1000  0.05  0.3   0.75  0.2   peak  1.0
#axis-t     bl    100   500   %1.0f       1

*end  # remove spectral traces

#################################################################
### Build the final display
#################################################################

*call image trace-image
rangen2     1

traces      traces.0
display                 wg
wiggle                  line              wiggle-line
wiggle                  fill-pos          trace-pos-fill
wiggle                  fill-neg          trace-neg-fill
```

```
traces      spectra.0
display                 wgva
wiggle                  line            spectrum-line
wiggle                  fill-pos        trace-pos-fill
#wiggle                 fill-neg        trace-neg-fill

traces      spcgram.0
#display                wg
display                 vi
wiggle                  line            wiggle-line
wiggle                  fill-pos        trace-pos-fill
wiggle                  fill-neg        trace-neg-fill
wiggle                  fill-pal        pal-ampl

object      button1
object      button2

### Start the viewer (or connect to a running viewer)

*call gui   gui-traces
trace-image

layout      horiz

dockers     # request docking windows
otree
prop
```

11.3 Appendix C. Hierarchy for IGeoS viewer class UI_X_Dvelmod

UI_X_SURFACE Struct Reference
[Core library]

`#include <ui_X_surface.h>`

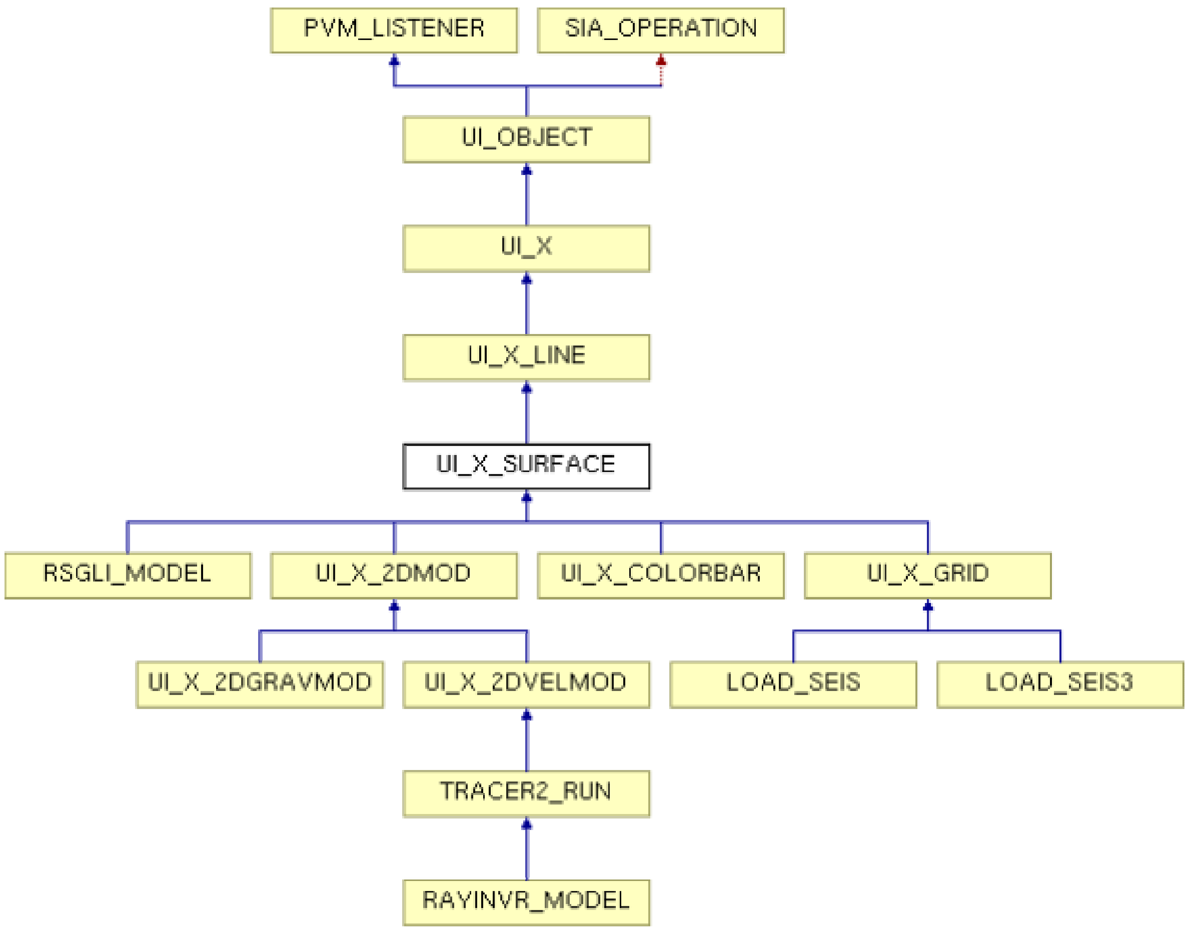


Figure 11.1 Inheritance diagram for UI_X_SURFACE

List of all members.

Detailed Description

Surface.

It can also be rendered as a line (wireframe)

Public Member Functions

	UI_X_SURFACE (const char *name=NULL, int **type**=UIX_SURFACE) *default constructor*

	UI_X_SURFACE (**RECORD_SURFACE** *r, **USER_MAP** *_ut, int **type**=UIX_SURFACE) *client*
	UI_X_SURFACE (**RECORD_VELMOD** *r, int **type**=UIX_2DVELMOD) *client*
	UI_X_SURFACE (int **tid**, int **tag**) *server-side*
	UI_X_SURFACE (int **tid**, int **tag**, int **type**) *server-side, without reading the object, for use in derived classes*
virtual **EVALTYPE**	**eval** (const char ***object**, const char *function, **AEVALTYPE** &params) *Evaluates a custom function named 'function' in 'object', with parameter 'params'.*
virtual int	**rendering_type** () const *type of the object used during rendering Not ethat all objects derived from this class are rendered as UIX_SURFACE*
virtual const float *	**matrix** () const *Returns the OpenGL matrix to use for this object.*
virtual const char *	**name** () const *Name for the image directory, etc.*
virtual const char *	**comment** () const *tool tip*
virtual const char *	**status** () const *status line displayed on MouseOver*
virtual const char *	**status** (const **UI_X_CURSOR** &p) const *Message appearing in the status bar when the cursor is pointing at the object.*
virtual **boolean**	**set_style** (const char *item, const char ***feature**, const char *name) *set custom line style*
virtual **boolean**	**begin_drawing** (int drawing) *Method called by the viewer when the drawing #'drawing' is about to start building.*

virtual void	**end_drawing** (int drawing, const char *error_message=NULL) *After the object is painted in GL, the xviewer uses this method to report success (if error_message=NULL) (or error otherwise) to the caller.*
virtual **boolean**	**call** (int **type**, int instruction, const char *param=NULL) *Interactive actions corresponding to **SIA_MODULE::call()**.*
virtual void	**clear_all** () *delete all segments of data points*
virtual **boolean**	**closed** () const *returns TRUE for a closed line (loop)*
virtual **boolean**	**constant_style** () const *returns TRUE if the style (color, lines) needs to be determined only once for the current line segment*
POINT3	**center** (int) const *Center of the grid (in object units; only one segment).*
virtual **boolean**	**segment** (int segm, int typ) *sets the segment number for the line-drawing functions*
virtual **boolean**	**point** (int ind, **POINT3** &vertex) *line or surface point ind of the current segment*
void	**point** (int ind, double &_x, double &_y, double &_z)
void	**triang** (int ind, **POINT3** &c1, **POINT3** &c2, **POINT3** &c3, int &fill) *returns corners of the triangle ind and its fill style for plotting*
virtual **POINT3**	**normal** (int **n**) *Following a call to triangle(.*
virtual const char *	**string** (int ind, **POINT3** &pos, **SIA_TEXT_STYLE** &style) *returns text string #'ind' that needs to be plotted on the object.*
SIA_RGB_COLOR const &	**fill_color** (int **type**=0) const *fill color for the current triangle (the one for which triangle(.*
float	**shininess** () const *returns shininess of the triangle called last*
virtual **ARRAY**< **UI_X_GRELEM** * >	**graphic** () const *returns pointers to all graphic elements (fonts, colors, palettes) used by the object*

virtual **boolean**	**pack** () *pack and unpack all data for* ***PVM***
virtual **boolean**	**unpack** ()
virtual **boolean**	**pack_create** () *pack and unpack data during initialization*
virtual **boolean**	**unpack_create** () *(with UIO_CREATE instruction)*
virtual **boolean**	**pack_params** (int item, int number) *pack object parameters (usually smaller blocks dependent on the context)*
virtual **boolean**	**unpack_params** () *unpack object parameters (usually smaller blocks dependent on the context)*
virtual void	**properties** () *rebuild property arrays*
virtual **boolean**	**properties** (const **UI_X_CURSOR** &cursor) *Build property list for cursor at the specified point.*
void	**send** (int signal=UIO_SET) *submit the edited data to the mirror*
virtual void	**print** () *debugging printout for xviewer*
virtual void	**set_line_style** (int **tag**) *set line style for the object*
virtual void	**set_fill_style** (int **tag**) *set fill style for the object*
virtual void	**set_palette** (int **tag**) *set color palette for the object*
virtual void	**set_font** (int **tag**) *set color palette for the object*
virtual void	**set_text_style** (int **tag**) *set color palette for the object*
void	**set_fill** ()

	Helper: set the internal fill style and color from the current value of _fill.
virtual **boolean**	**mouse_action** (const **MOUSE_EVENT** &cursor) *passes to the image mouse pressed event.*

Public Attributes

USER_MAP *	**ut** *the surface table (MAP)*
AHEADER	**x**
AHEADER	**y**
AHEADER	**z** *coordinates in the table to plot*
AHEADER	**lp** *palette argument parameter for paletted lines*
AHEADER	**fp** *palette argument parameter for paletted fills*
CHARSTR	**_name** *name of the object*
REAL_EDIT	**gain** *additional gain applied to palette fills*

Protected Member Functions

virtual **boolean**	**_build** (const **POINT3** &box_min, const **POINT3** &box_max, double res) *set sampling box and resolution and rebuild this object only (without children)*
void	**offsets** (const **AHEADER** &h, **byte** *&**b**, **_LONG** &o1, **_LONG** &o2) ***Set** record start and byte offsets for field h.*
void	**paletted_fill** (double level) *Sets custom fill color based on 'level'.*
virtual **POINT3**	**normal** (int ix, int iy) const

	Returns unit normal vector at node (ix,iy).
void	**interp** (const **POINT3** &**p11**, const **POINT3** &**p12**, const **POINT3** &**p22**, const **POINT3** &**p21**, double **x**, double **y**, **POINT3** &v) *interpolate the values at (x,y) from p11, p12, p22, p21, assuming they ar taken at (0,_split) intervals*
void	**interp** (double **x**, double **y**, **SIA_MATERIAL** &m) *similar to* ***interp()*** *- interpolate material properties*
virtual void	**precompute** (int i1, int i2) *compute p11,p12,p21,p22 and the corresponding m's*

Protected Attributes

int	**_n1**
int	**_n2** *dimensions of the current grid*
int	**_skip1**
int	**_skip2** *numbers of skipped grids points while rendering*
int	**_n1s**
int	**_n2s** *numbers of plotted points along _n1 and _n2*
int	**_i1**
int	**_i2**
int	**_it** *current cell index*
_LONG	**_ox1**
_LONG	**_ox2**
_LONG	**_oy1**
_LONG	**_oy2**
_LONG	**_oz1**
_LONG	**_oz2** *byte offsets for coordinates*

_LONG	**_olp1**
_LONG	**_olp2**
_LONG	**_ofp1**
_LONG	**_ofp2**
POINT3	**ex**
POINT3	**ey**
POINT3	**ez** *directional vectors for mapping (x,y,z)*
byte *	**_rx**
byte *	**_ry**
byte *	**_rz** *records of the fields*
byte *	**_rfp**
byte *	**_rlp** *palette parameter records*
FILL_STYLE_EDIT	**_fill** *fill style tag*
PALETTE_EDIT	**_fill_palette** *optional palette tag used for fill styles*
SIA_FILL_STYLE *	**_fs** *current fill style*
SIA_COLOR_PALETTE *	**_palf** *Pointer to the actually used fill palette.*
INT_EDIT	**_split** *If >= 0, this is the number of subdivisions of the smallest cell during rendering.*
int	**cell1**
int	**cell2** *subindexes of the interpolation cell*
POINT3	**p11**
POINT3	**p12**

POINT3	**p21**
POINT3	**p22** *cell corners for interpolation of the current segment*
SIA_MATERIAL	**m11**
SIA_MATERIAL	**m12**
SIA_MATERIAL	**m21**
SIA_MATERIAL	**m22** *material at cell corners for interpolation of the current segment*
UI_X_PROPERTY_EDIT	**info** [2] *non-editable properties*

Friends

class	**RECORD_SURFACE**
class	**RECORD_VELMOD**
class	**RECORD_GRID_TABLE**
class	**UIP_X_TRACES**
class	**RECORD_COLORBAR**

Constructor & Destructor Documentation

```
UI_X_SURFACE::UI_X_SURFACE ( const char *  name = NULL,
                             int           type = UIX_SURFACE
                           )
```

default constructor

```
UI_X_SURFACE::UI_X_SURFACE ( RECORD_SURFACE *  r,
                             USER_MAP *        _ut,
                             int               type = UIX_SURFACE
                           )
```

client

UI_X_SURFACE::UI_X_SURFACE (RECORD_VELMOD * *r*,
int *type* = `UIX_2DVELMOD`
)

client

UI_X_SURFACE::UI_X_SURFACE (int *tid*,
int *tag*
)

server-side

UI_X_SURFACE::UI_X_SURFACE (int *tid*,
int *tag*,
int *type*
)

server-side, without reading the object, for use in derived classes

Member Function Documentation

virtual EVALTYPE UI_X_SURFACE::eval (const char * *object*,
const char * *function*,
AEVALTYPE & *params*
) **`[inline, virtual]`**

Evaluates a custom function named 'function' in 'object', with parameter 'params'.

Reimplemented from **UI_X_LINE**.

Reimplemented in **RSGLI_MODEL**.

virtual int UI_X_SURFACE::rendering_type ( ) const `[inline, virtual]`

type of the object used during rendering Not ethat all objects derived from this class are rendered as UIX_SURFACE

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_2DVELMOD**, and **UI_X_2DGRAVMOD**.

virtual const float* UI_X_SURFACE::matrix ( ) const [inline, virtual]

Returns the OpenGL matrix to use for this object.

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**.

virtual const char* UI_X_SURFACE::name ( ) const [virtual]

Name for the image directory, etc.

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, **LOAD_SEIS**, **LOAD_SEIS3**, **RAYINVR_MODEL**, **RSGLI_MODEL**, and **TRACER2_RUN**.

virtual const char* UI_X_SURFACE::comment ( ) const [virtual]

tool tip

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, and **RAYINVR_MODEL**.

virtual const char* UI_X_SURFACE::status ( ) const [virtual]

status line displayed on MouseOver

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, and **RAYINVR_MODEL**.

virtual const char* UI_X_SURFACE::status (const UI_X_CURSOR & *p*) `const [inline, virtual]`

Message appearing in the status bar when the cursor is pointing at the object.

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DMOD**, and **UI_X_2DVELMOD**.

virtual boolean UI_X_SURFACE::set_style (const char * ***item,***
const char * ***feature,***
const char * ***name***
) `[virtual]`

set custom line style

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_2DMOD**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, and **RSGLI_MODEL**.

virtual boolean UI_X_SURFACE::begin_drawing (int ***drawing*** **)** `[inline, virtual]`

Method called by the viewer when the drawing #'drawing' is about to start building.

If this method returns FALSE, the image is not drawn.

virtual void UI_X_SURFACE::end_drawing (int ***drawing,***
const char * ***error_message =*** `NULL`
) `[inline, virtual]`

After the object is painted in GL, the xviewer uses this method to report success (if error_message=NULL) (or error otherwise) to the caller.

Reimplemented from **UI_X_LINE**.

virtual boolean UI_X_SURFACE::call (int ***type,***
int ***instruction,***

const char * *param* = `NULL`

) `[virtual]`

Interactive actions corresponding to **SIA_MODULE::call()**.

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **RAYINVR_MODEL**, and **RSGLI_MODEL**.

virtual void UI_X_SURFACE::clear_all ( ) `[virtual]`

delete all segments of data points

Reimplemented from **UI_X_LINE**.

virtual boolean UI_X_SURFACE::closed ( ) const `[inline, virtual]`

returns TRUE for a closed line (loop)

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_2DVELMOD**, and **UI_X_2DGRAVMOD**.

virtual boolean UI_X_SURFACE::constant_style ( ) const `[virtual]`

returns TRUE if the style (color, lines) needs to be determined only once for the current line segment

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, and **RAYINVR_MODEL**.

POINT3 UI_X_SURFACE::center (int) const `[inline, virtual]`

Center of the grid (in object units; only one segment).

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_2DVELMOD**, **UI_X_GRID**, and **RAYINVR_MODEL**.

virtual boolean UI_X_SURFACE::segment (int ***segm,***
int ***typ***
) `[virtual]`

sets the segment number for the line-drawing functions

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **LOAD_SEIS3**, **RAYINVR_MODEL**, **RSGLI_MODEL**, and **TRACER2_RUN**.

virtual boolean UI_X_SURFACE::point (int ***ind,***
POINT3 & ***vertex***
) `[virtual]`

line or surface point ind of the current segment

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, **RAYINVR_MODEL**, **RSGLI_MODEL**, and **TRACER2_RUN**.

void UI_X_SURFACE::point (int ***ind,***
double & ***_x,***
double & ***_y,***
double & ***_z***
)

Reimplemented in **RAYINVR_MODEL**.

void UI_X_SURFACE::triang (int ***ind,***
POINT3 & ***c1,***
POINT3 & ***c2,***
POINT3 & ***c3,***
int & ***fill***

)

returns corners of the triangle ind and its fill style for plotting

virtual POINT3 UI_X_SURFACE::normal (int *n*) [virtual]

Following a call to triangle(.

..), or **polygon()**, this method returns the directional cosines (unit vectors normal to the surface) at its n-th vertex

Reimplemented from **UI_X_LINE**.

virtual const char* UI_X_SURFACE::string (int *ind*, POINT3 & *pos*, SIA_TEXT_STYLE & *style*) [inline, virtual]

returns text string #'ind' that needs to be plotted on the object.

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, **LOAD_SEIS3**, **RAYINVR_MODEL**, and **RSGLI_MODEL**.

SIA_RGB_COLOR const& UI_X_SURFACE::fill_color (int *type* = *0*) const

fill color for the current triangle (the one for which triangle(.

..) was called most recently). 'type' specified the type of color: 0 - the "emitted" light; 1 - response to ambient light; 2 - response to diffuse light; 3 - response to specular light;

float UI_X_SURFACE::shininess ( ) const

returns shininess of the triangle called last

virtual ARRAY<UI_X_GRELEM*> UI_X_SURFACE::graphic ( ) const [virtual]

returns pointers to all graphic elements (fonts, colors, palettes) used by the object

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, **RAYINVR_MODEL**, and **RSGLI_MODEL**.

virtual boolean UI_X_SURFACE::pack ( ) [virtual]

pack and unpack all data for **PVM**

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, **LOAD_SEIS**, **LOAD_SEIS3**, **RAYINVR_MODEL**, and **RSGLI_MODEL**.

virtual boolean UI_X_SURFACE::unpack ( ) [virtual]

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, **LOAD_SEIS**, **LOAD_SEIS3**, **RAYINVR_MODEL**, and **RSGLI_MODEL**.

virtual boolean UI_X_SURFACE::pack_create ( ) [virtual]

pack and unpack data during initialization

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, **LOAD_SEIS**, **LOAD_SEIS3**, **RAYINVR_MODEL**, **RSGLI_MODEL**, and **TRACER2_RUN**.

virtual boolean UI_X_SURFACE::unpack_create ( ) [virtual]

(with UIO_CREATE instruction)

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, **LOAD_SEIS**, **LOAD_SEIS3**, **RAYINVR_MODEL**, **RSGLI_MODEL**, and **TRACER2_RUN**.

virtual boolean UI_X_SURFACE::pack_params (int *item*,
int *number*
) [inline, virtual]

pack object parameters (usually smaller blocks dependent on the context)

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_GRID**, and **RSGLI_MODEL**.

virtual boolean UI_X_SURFACE::unpack_params ( ) `[inline, virtual]`

unpack object parameters (usually smaller blocks dependent on the context)

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_GRID**, and **RSGLI_MODEL**.

virtual void UI_X_SURFACE::properties ( ) `[virtual]`

rebuild property arrays

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, **LOAD_SEIS**, **LOAD_SEIS3**, **RAYINVR_MODEL**, and **RSGLI_MODEL**.

virtual boolean UI_X_SURFACE::properties (const UI_X_CURSOR & *cursor*) `[inline, virtual]`

Build property list for cursor at the specified point.

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, and **UI_X_GRID**.

void UI_X_SURFACE::send (int *signal* = *`UIO_SET`*) `[inline, virtual]`

submit the edited data to the mirror

Reimplemented from **UI_X_LINE**.

virtual void UI_X_SURFACE::print ( ) `[virtual]`

debugging printout for xviewer

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, **UI_X_GRID**, **LOAD_SEIS3**, **RAYINVR_MODEL**, and **TRACER2_RUN**.

virtual void UI_X_SURFACE::set_line_style (int *tag*) [inline, virtual]

set line style for the object

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_2DVELMOD**, and **UI_X_2DGRAVMOD**.

virtual void UI_X_SURFACE::set_fill_style (int *tag*) [inline, virtual]

set fill style for the object

Reimplemented from **UI_X**.

Reimplemented in **UI_X_2DVELMOD**, and **UI_X_2DGRAVMOD**.

virtual void UI_X_SURFACE::set_palette (int *tag*) [inline, virtual]

set color palette for the object

Reimplemented from **UI_X**.

Reimplemented in **UI_X_2DVELMOD**, and **UI_X_2DGRAVMOD**.

virtual void UI_X_SURFACE::set_font (int *tag*) [inline, virtual]

set color palette for the object

Reimplemented from **UI_X**.

Reimplemented in **UI_X_2DVELMOD**, and **UI_X_2DGRAVMOD**.

virtual void UI_X_SURFACE::set_text_style (int *tag*) [inline, virtual]

set color palette for the object

Reimplemented from **UI_X_LINE**.

void UI_X_SURFACE::set_fill ( )

Helper: set the internal fill style and color from the current value of _fill.

virtual boolean UI_X_SURFACE::mouse_action (const MOUSE_EVENT & *cursor*) `[inline, virtual]`

passes to the image mouse pressed event.

Returns TRUE if the image has handled the event or FALSE if it should be interpreted by the viewer. The default **UI_X** behavior is to pass the event to the sub-objects.

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, and **RAYINVR_MODEL**.

virtual boolean UI_X_SURFACE::_build (const POINT3 & *box_min*, const POINT3 & *box_max*, double *res*) `[protected, virtual]`

set sampling box and resolution and rebuild this object only (without children)

Reimplemented from **UI_X_LINE**.

Reimplemented in **UI_X_COLORBAR**, **UI_X_2DVELMOD**, **UI_X_2DGRAVMOD**, and **LOAD_SEIS3**.

void UI_X_SURFACE::offsets (const AHEADER & *h*, byte *& *b*, _LONG & *o1*, _LONG & *o2*) `[protected]`

Set record start and byte offsets for field h.

void UI_X_SURFACE::paletted_fill (double *level*) `[protected]`

Sets custom fill color based on 'level'.

virtual POINT3 UI_X_SURFACE::normal (int ***ix*****,**
int ***iy***
) **`const [protected, virtual]`**

Returns unit normal vector at node (ix,iy).

The vector is orthogonal to the surface after scaling into GL space.

Reimplemented in **UI_X_GRID**.

void UI_X_SURFACE::interp (const POINT3 & ***p11*****,**
const POINT3 & ***p12*****,**
const POINT3 & ***p22*****,**
const POINT3 & ***p21*****,**
double ***x*****,**
double ***y*****,**
POINT3 & ***v***
) **`[protected]`**

interpolate the values at (x,y) from p11, p12, p22, p21, assuming they ar taken at (0,_split) intervals

void UI_X_SURFACE::interp (double ***x*****,**
double ***y*****,**
SIA_MATERIAL & ***m***
) **`[protected]`**

similar to **interp()** - interpolate material properties

virtual void UI_X_SURFACE::precompute (int ***i1*****,**

int ***i2***

) `[protected, virtual]`

compute p11,p12,p21,p22 and the corresponding m's

Reimplemented in **UI_X_GRID**, and **LOAD_SEIS3**.

Friends And Related Function Documentation

friend class RECORD_SURFACE `[friend]`

Reimplemented from **UI_X_LINE**.

friend class RECORD_VELMOD `[friend]`

friend class RECORD_GRID_TABLE `[friend]`

friend class UIP_X_TRACES `[friend]`

friend class RECORD_COLORBAR `[friend]`

Member Data Documentation

USER_MAP* UI_X_SURFACE::ut

the surface table (MAP)

AHEADER UI_X_SURFACE::x

AHEADER UI_X_SURFACE::y

AHEADER UI_X_SURFACE::z

coordinates in the table to plot

AHEADER UI_X_SURFACE::lp

palette argument parameter for paletted lines

AHEADER UI_X_SURFACE::fp

palette argument parameter for paletted fills

CHARSTR UI_X_SURFACE::_name

name of the object

REAL_EDIT UI_X_SURFACE::gain

additional gain applied to palette fills

int UI_X_SURFACE::_n1 **`[protected]`**

int UI_X_SURFACE::_n2 **`[protected]`**

dimensions of the current grid

int UI_X_SURFACE::_skip1 **`[protected]`**

int UI_X_SURFACE::_skip2 **`[protected]`**

numbers of skipped grids points while rendering

int UI_X_SURFACE::_n1s **`[protected]`**

int UI_X_SURFACE::_n2s **`[protected]`**

numbers of plotted points along _n1 and _n2

int UI_X_SURFACE::_i1 **`[protected]`**

int UI_X_SURFACE::_i2 **`[protected]`**

int UI_X_SURFACE::_it **`[protected]`**

current cell index

_LONG UI_X_SURFACE::_ox1 **`[protected]`**

_LONG UI_X_SURFACE::_ox2 **`[protected]`**

_LONG UI_X_SURFACE::_oy1 **`[protected]`**

_LONG UI_X_SURFACE::_oy2 **`[protected]`**

_LONG UI_X_SURFACE::_oz1 `[protected]`

_LONG UI_X_SURFACE::_oz2 `[protected]`

byte offsets for coordinates

_LONG UI_X_SURFACE::_olp1 `[protected]`

_LONG UI_X_SURFACE::_olp2 `[protected]`

_LONG UI_X_SURFACE::_ofp1 `[protected]`

_LONG UI_X_SURFACE::_ofp2 `[protected]`

POINT3 UI_X_SURFACE::ex `[protected]`

POINT3 UI_X_SURFACE::ey `[protected]`

POINT3 UI_X_SURFACE::ez `[protected]`

directional vectors for mapping (x,y,z)

byte* UI_X_SURFACE::_rx
`[protected]`

byte * UI_X_SURFACE::_ry `[protected]`

byte * UI_X_SURFACE::_rz `[protected]`

records of the fields

byte * UI_X_SURFACE::_rfp `[protected]`

byte * UI_X_SURFACE::_rlp
`[protected]`

palette parameter records

FILL_STYLE_EDIT UI_X_SURFACE::_fill `[protected]`

fill style tag

PALETTE_EDIT UI_X_SURFACE::_fill_palette `[protected]` optional palette tag used for fill styles

SIA_FILL_STYLE* UI_X_SURFACE::_fs `[protected]`

current fill style

SIA_COLOR_PALETTE* UI_X_SURFACE::_palf `[protected]`

Pointer to the actually used fill palette.

INT_EDIT UI_X_SURFACE::_split `[protected]`

If >= 0, this is the number of subdivisions of the smallest cell during rendering.

If < 0, "pixel registration" is used, with painted cells centered on the map points.

int UI_X_SURFACE::cell1 `[protected]`

int UI_X_SURFACE::cell2 `[protected]`

subindexes of the interpolation cell

POINT3 UI_X_SURFACE::p11 `[protected]`

POINT3 UI_X_SURFACE::p12 `[protected]`

POINT3 UI_X_SURFACE::p21 `[protected]`

POINT3 UI_X_SURFACE::p22 `[protected]`

cell corners for interpolation of the current segment

SIA_MATERIAL UI_X_SURFACE::m11 `[protected]`

SIA_MATERIAL UI_X_SURFACE::m12 `[protected]`

SIA_MATERIAL UI_X_SURFACE::m21 `[protected]`

SIA_MATERIAL UI_X_SURFACE::m22 `[protected]`

material at cell corners for interpolation of the current segment

UI_X_PROPERTY_EDIT UI_X_SURFACE::info[2] `[protected]` non-editable properties

---

The documentation for this struct was generated from the following file:

- include/**ui_X_surface.h**