Colada

Release 0.1.0

Kerim Khemraev

Oct 18, 2023

CONTENTS

This documentation contains information mostly about upper layer of the platform. To see the fundamential documentation please refer to the Slicer 3D documentation.

CHAPTER ONE

OVERVIEW

Colada is a crossplatform desktop software that utilizes full power of C++, Python and Julia to make geological/geophysical data processing effective, fast and robust. Currently it only supports:

1) Data visualization (including pre/post stack seismic);
2) Various operations on data (like transforms, interpolation, interactive tools etc.);
3) 2D/3D seismic wavemodeling including scalar, acoustic, elastic, VTI/TTI models (Unix only);
4) 2D/3D Full Waveform Inversion (FWI), Reverse Time Migration (RTM), Least-Squares Reverse Time Migration (LSRTM) (Unix only);

Most functionality is implemented through extensions and modules. Modules are written on either C++ or Python. From the user side there is no any visible difference between C++ and Python modules. On the other hand use of Python ultimately reduces time of new functionality implementation. Many open source projects are involved and more are yet going to be involved.

The first great thing about Colada is that it has all the modern visualization capabilities. It is not only about 2D/3D visualization that is ready to use but also the functionality that will be implemented in the near future like: Virtual Reality, Holographic Display and 3D printing.

The second great thing is that even the core is written in C++ all the functionality is available from embedded Python interpreter. Even Graphical User Interface! That grants almost unlimited abilities to the user with Python knowledge. Reading low-popular files, processing multiples files in loop without monotonic routine or adding new functionality is as simple as you familiar with Python.

The third great thing is about large-scale problems. For such tasks there is Julia interpreter wich allows to write code as fast as using Python and be as efficient as C language. Best choice for mathematicians.

All of this is based on core with over 20 years of history Slicer 3D and the community that supports it.

CHAPTER

TWO

GETTING STARTED

This page contains information that you need to get started with Colada, including how to install and use basic features and where to find more information.

2.1 System requirements

Colada runs on any Windows, Mac, or Linux computer that was released in the last 5 years. Older computers may work (depending mainly on graphics capabilities).

2.1.1 Operating system versions

- Windows: Windows 10 or 11, with all recommended updates installed. Windows 10 Version 1903 (May 2019 Update) version or later is required for support of international characters (UTF-8) in filenames and text. Microsoft does not support Windows 8.1 and Windows 7 anymore and Colada is not tested on these legacy operating system versions, but may still work.
- macOS: macOS High Sierra (10.13) or later (both Intel and ARM based systems). Latest public release is recommended.
- Linux: Ubuntu 18.04 or laterCentOS 7 or later. Latest LTS (Long-term-support) version is recommended.

2.1.2 Recommended hardware configuration

- Memory: more than 4GB (8 or more is recommended). As a general rule, have 10x more memory than the amount of data that you load.
- Display: a minimum resolution of 1024 by 768 (1280 by 1024 or better is recommended).
- Graphics: Dedicated graphics hardware (discrete GPU) memory is recommended for fast volume rendering. GPU: Graphics must support minimum OpenGL 3.2. Integrated graphics card is sufficient for basic visualization. Discrete graphics card (such as NVidia GPU) is recommended for interactive 3D volume rendering and fast rendering of complex scenes. GPU texture memory (VRAM) should be larger than your largest dataset (e.g., working with 2GB data, get VRAM > 4GB) and check that your images fit in maximum texture dimensions of your GPU hardware. Except rendering, most calculations are performed on CPU, therefore having a faster GPU will generally not impact the overall speed of the application.
- Some computations in Colada are multi-threaded and will benefit from multi core, multi CPU configurations.
- Interface device: a three button mouse with scroll wheel is recommended. Pen, multi-touchscreen, touchpad, and graphic tablet are supported. All OpenVR-compatible virtual reality headsets are supported for virtual reality display.

- Internet connection to access extensions, Python packages, online documentation, sample data sets, and tutorials.

Once downloaded, follow the instructions below to complete installation:

2.1.3 Windows

- Run the installer.
  - Current limitation: Installation path must only contain English (ASCII printable) characters because otherwise some Python packages may not load correctly (see this issue for more details).
- Run Colada from the Windows start menu.
- Use "Apps & features" in Windows settings to remove the application.

2.1.4 Mac

- Open the install package (.dmg file).
- Drag the Colada application (Colada.app) to your Applications folder (or other location of your choice).
  - This step is necessary because content of a .dmg file is opened as a read-only volume, and you cannot install extensions or Python packages into a read-only volume.
- Delete the Colada.app folder to uninstall.

Note: currently Colada packages are not signed. Therefore, when the application is started the first time the following message is displayed: "Colada… can't be opened because it is from an unidentified developer". To resolve this error, locate the application in Finder and right-click (two-finger click) and click `Open`. When it says `This app can't be opened` go ahead and hit cancel. Right click again and say `Open` (yes, you need to repeat the same as you did before - the outcome will be different than the first time). Click the `Open` (or `Open anyway`) button to start the application. See more explanation and alternative techniques here.

2.1.5 Linux

- Open the tar.gz archive and copy directory to the location of your choice.
- Installation of additional packages may be necessary depending on the Linux distribution and version, as described in subsections below.
- Run the `Colada` executable.
- Remove the directory to uninstall.

Notes:

- Colada is expected to work on the vast majority of desktop and server Linux distributions. The system is required to provide at least GLIBC 2.17 and GLIBCCC 3.4.19. For more details, read here.
- Getting command-line arguments and process output containing non-ASCII characters requires the system to use a UTF-8 locale. If the system uses a different locale then the `export LANG="C.UTF-8"` command may be used before launching the application to switch to an acceptable locale.

Debian / Ubuntu

The following may be needed on fresh debian or ubuntu:

```
sudo apt-get install libpulse-dev libnss3 libglu1-mesa
sudo apt-get install --reinstall libxcb-xinerama0
```

Warning: Debian 10.12 users may encounter an error when launching Colada:

```
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to␣
↪run on Wayland anyway.
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was␣
↪found.
This application failed to start because no Qt platform plugin could be initialized.␣
↪Reinstalling the application may fix this problem.

Available platform plugins are: xcb.
```

The solution is to create symlink:

```
sudo ln -s /usr/lib/x86_64-linux-gnu/libxcb-util.so /usr/lib/x86_64-linux-gnu/libxcb-
↪util.so.1
```

Fedora

Install the dependencies:

```
sudo dnf install mesa-libGLU libnsl
```

The included libcrypto.so.1.1 in the Colada installation is incompatible with the system libraries used by Fedora 35. The fix, until it is updated, is to move/remove the included libcrypto files:

```
$COLADA_ROOT/lib/Colada-x.xx/libcrypto.*
```

2.2 Using Colada

Colada offers lots of features and gives users great flexibility in how to use them. As a result, new users may be overwhelmed with the number of options and have difficulty figuring out how to perform even simple operations. This is normal and many users successfully crossed this difficult stage by investing some time into learning how to use this software.

How to learn Colada?

2.2.1 Quick start

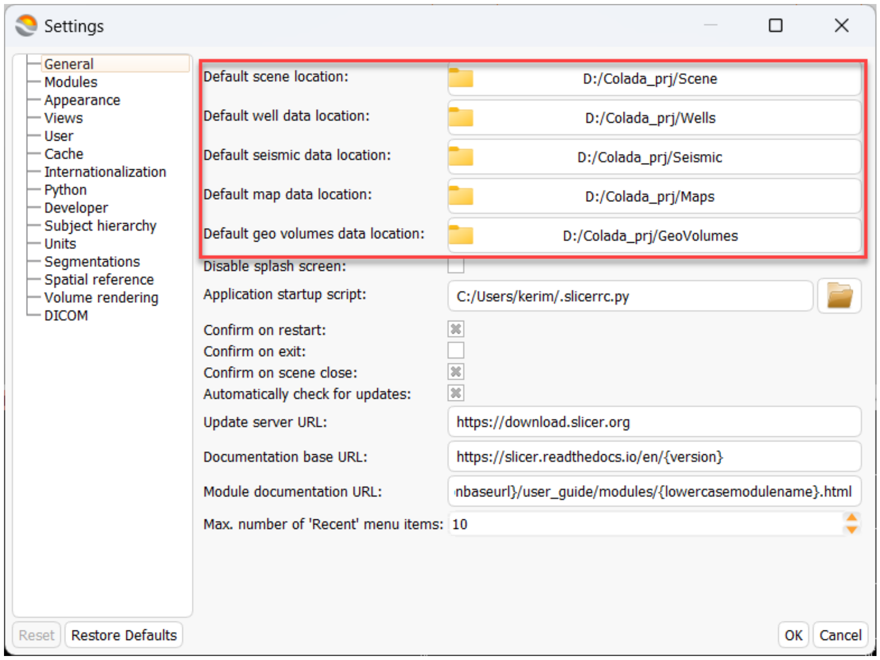
The easiest way to start is to open `Welcome to Colada` module and follow the proposed actions.

Note: When Colada is just installed and run for first time don't forget to run: `colada.init_python_julia_packages()`. This will install some packages that is required by some modules (Internet connection is required).

Setting default directories

Many different fileformats supported by VTK may be loaded to Colada as is.

Nevertheless geological data must be stored in HDF5 containers using structures defined by h5geo library. All the necessary information about spatial reference of data, units (length, temporal and angular), domain (TWT, OWT, TVD, TVDSS) reside in HDF5 containers as defined by h5geo. Each type of container (Seismic, Geo-Volumes, Wells, Maps) is stored in separate folders. This makes convenient to load/store data.

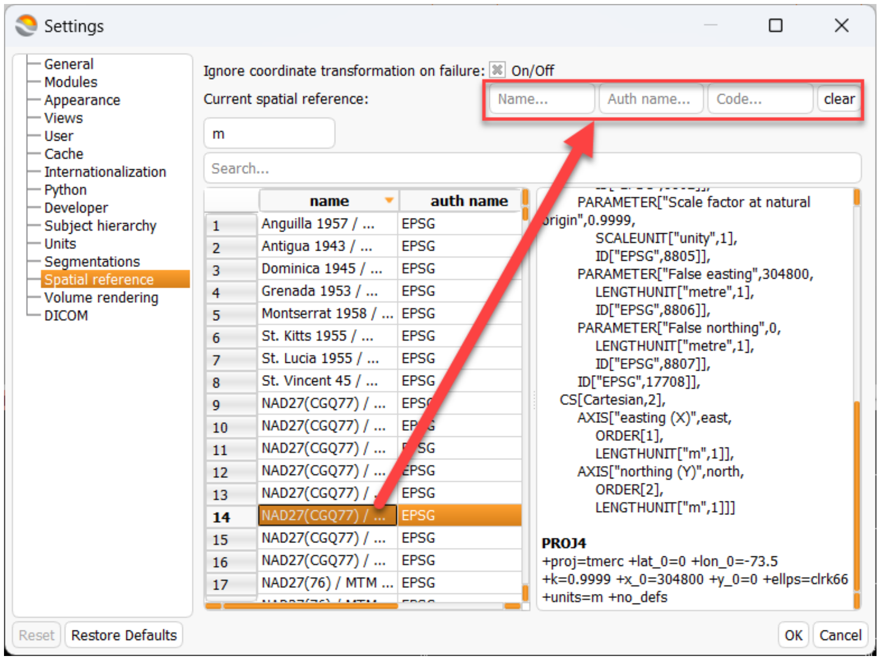


Setting spatial reference

The user is able to set spatial reference and units of the current session. All the geological data loaded to Colada will be transformed to spatial reference and units of the session.

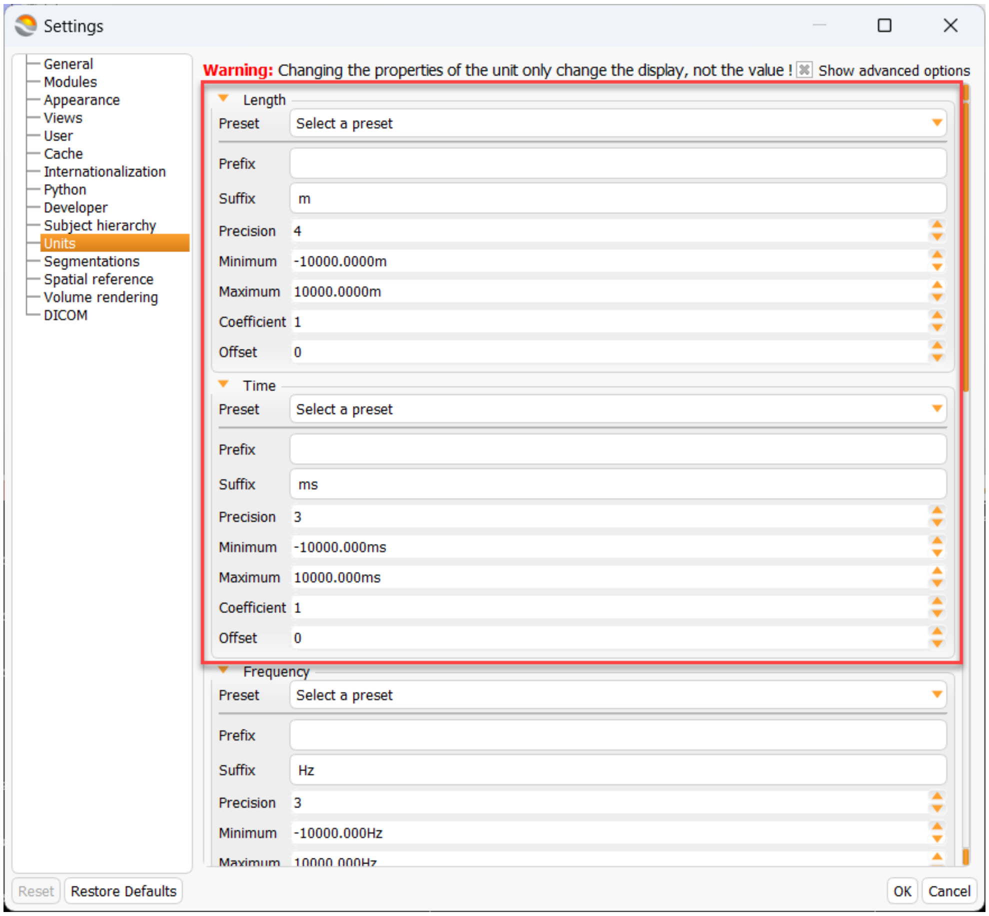
To change spatial reference find one in the table and using drag&drop technique place it to input widget like shown in the picture.

Note: It is recommended to work within some spatial reference. If one doesn't want to use it then don't forget to turn on the checkbox `Ignore coordinate transformation on failure` in `Toolbar menu->Edit->Application Settings->Spatial reference`. Though the units must be set anyway.



Setting units

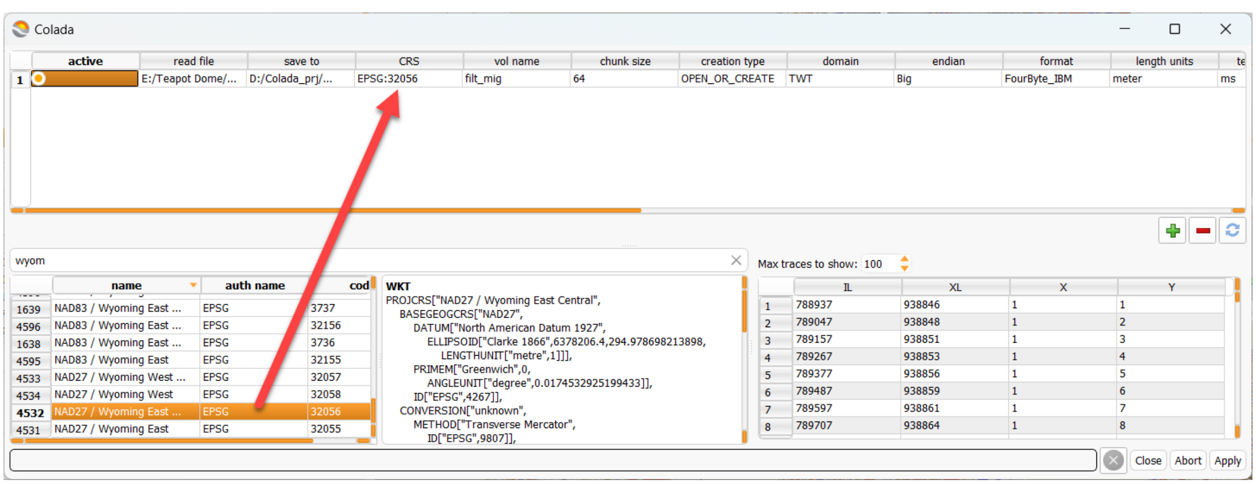
The most important is to set length and time units.



Read data

All readers are structured in similar way: there is a table with data to be read and table with allowed spatial references.

For example SEGY stack reader looks like shown in the picture.



A number of columns must be fille to read such data. Most of them are well known to seismic specialists. And all other are pretty self-described. To change spatial reference choose one from the table below and put it to the table above at specified row.

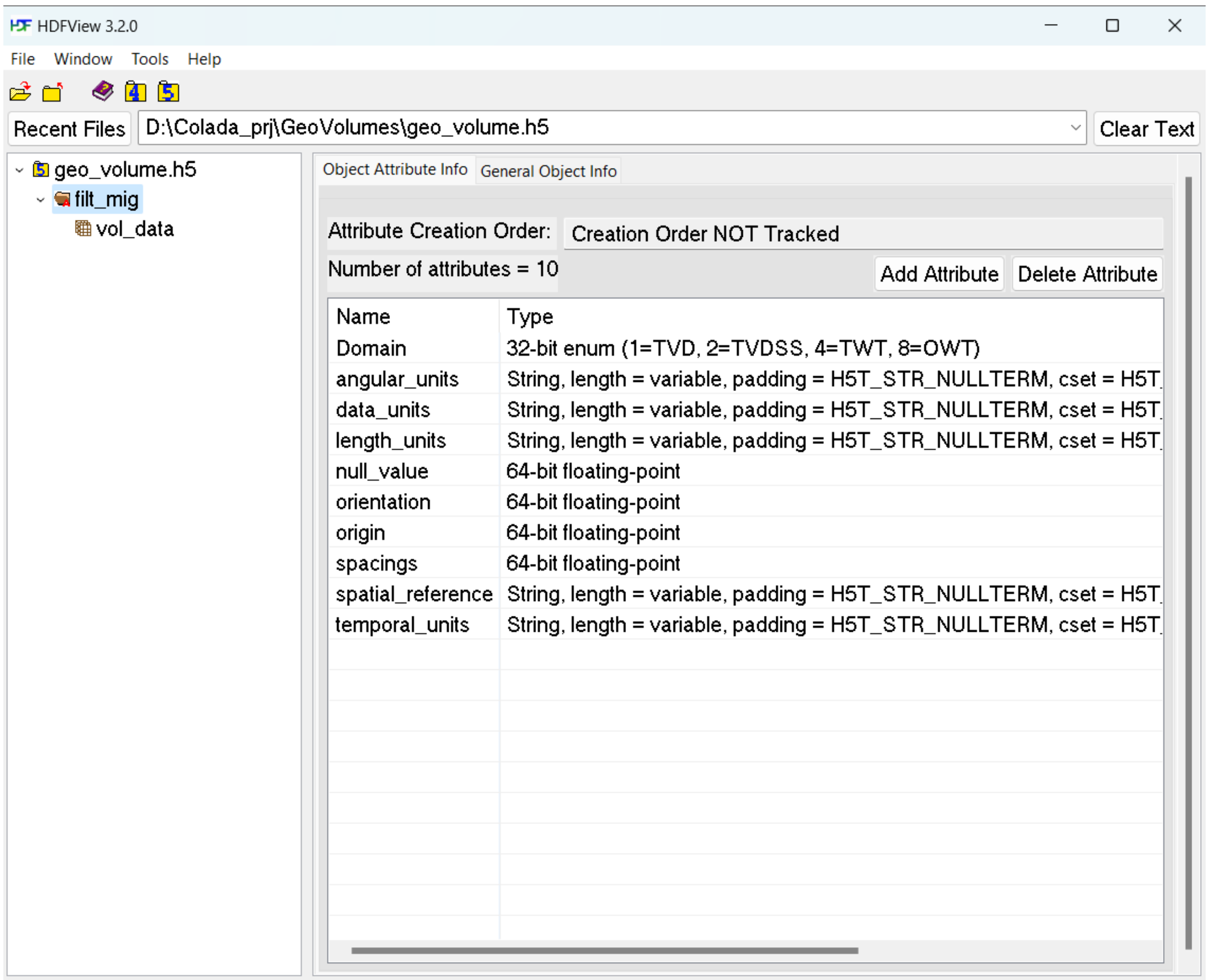
To update/remove row select that row and click on appropriate button.

View data

After the file is read it can be processed in Colada. But first it recommended to download and install HDFVIEW application. This application allows to view content on HDF5 file. It is good to understand `h5geo` data structures. But don't forget to close the container if it is going to be used by Colada with file locking set. Conainer becomes incaccessible if it is opened in HDFVIEW

> **Warning:** HDF5 file locking prevents user from data corruption. It is not recommended to avoid this setting as data may be lost if multiple processes write data to the same object at the same time. If one doesn't want to use it, open `ColadaLauncherSettings.ini` and set `HDF5_USE_FILE_LOCKING=FALSE`.

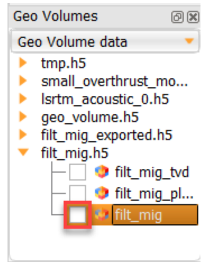
Here is a picture of how geo-volume look using HDFVIEW:



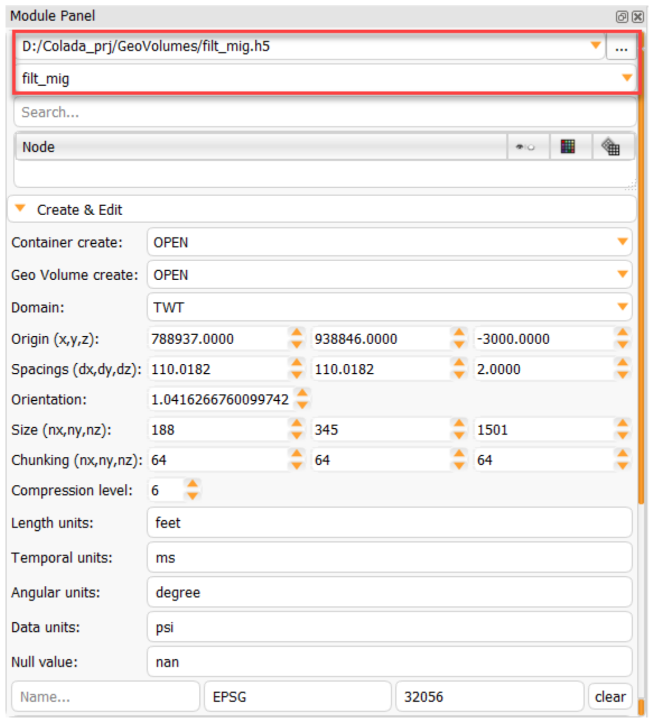
As one can see all the information is described by object's attributes.

The object may be viewed in Colada as well.

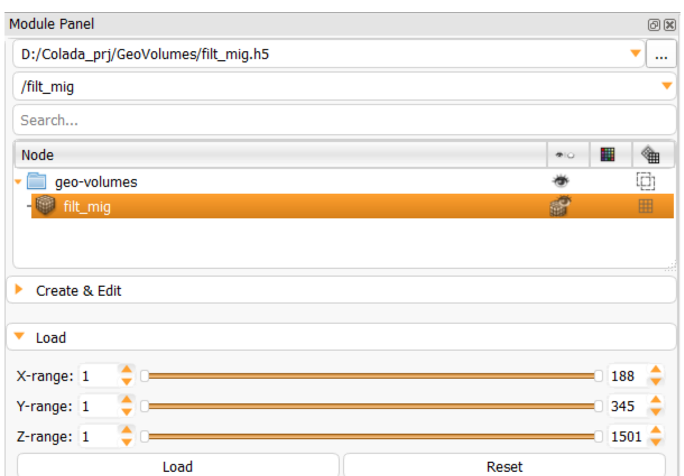
For that do the following: click on the checkbox in the treeview, current module will be changed to `GeoVolumes` and there in `Create & Edit` section one can find and edit information about this object.



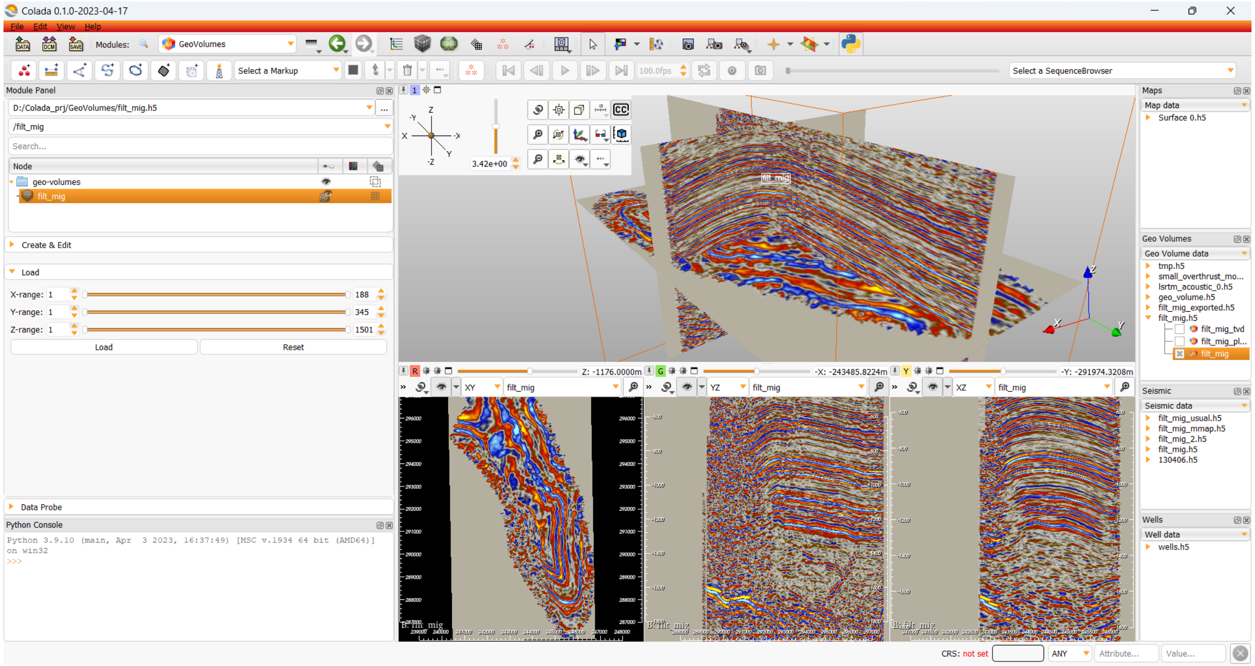
The same effect can be achieved by drag&drop item from treeview and putting it to the input boxes of the module.



To load the data select the range of XYZ and click `Load` button. Then an item will be appeared in the subject-hierarchy section. Click on **eye** to display the selected volume.



This will cause to display volume in the 2D slices. To display it on 3D view click on **eye** in each slice.



All the loaded data can be viewed in `Data` module.

Control how volume is visualized

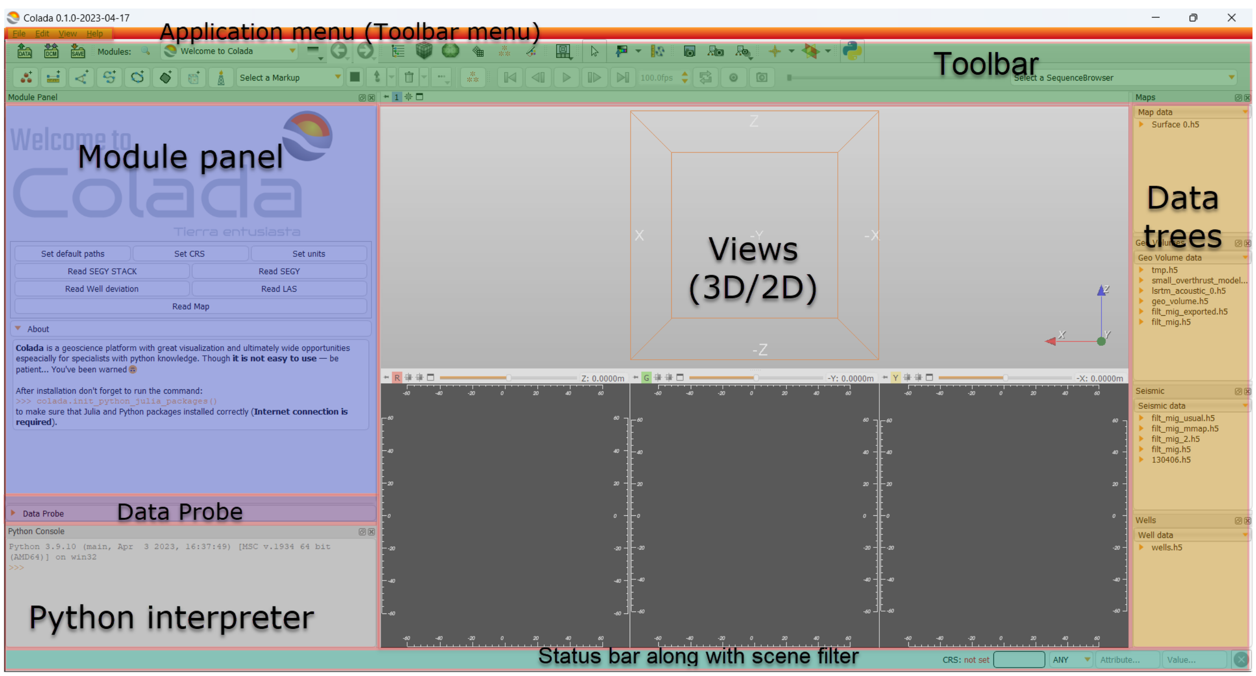
Go to the `Volumes` module, set current volume and modify whatever you want.

USER INTERFACE

3.1 Application overview

Colada stores all loaded data in a data repository, called the "scene" (or Colada scene or MRML scene). Each data set, such as an image volume, surface model, or point set, is represented in the scene as a "node".

Colada provides a large number "modules", each implementing a specific set of functions for creating or manipulating data in the scene. Modules typically do not interact with each other directly: they just all operate on the data nodes in the scene.



3.1.1 Module Panel

This panel (located by default on the left side of the application main window) displays all the options and features that the current module offers to the user. Current module can be selected using the **Module Selection** toolbar.

Data Probe

Data Probe is located at the bottom of the module panel. It displays information about view content at the position of the mouse pointer:

- Slice view information (displayed when the mouse is over a slice view):
  - Slice view name: `Red`, `Green`, `Yellow`, etc.
  - Three coordinate values, prefixed with `+-X,+-Y,+-Z`.
  - View orientation: `XY`, `YZ`, `XZ` and `Reformat` for any other orientation.
  - Slice spacing: distance between slices in this orientation.
- Volume layer information: three lines, one for each volume layer
  - Layer type: `L` (label), `F` (foreground), `B` (background).
  - Volume name, or `None` if no volume is selected for that layer.
  - Volume voxel (IJK) coordinates.
  - Voxel value. For label volumes the label name corresponding to the voxel value is also displayed.
- Segmentation information: for each segmentation visible at that position
  - Layer type: `S` (segmentation)
  - Segmentation name.
  - Segment names. Multiple segment names are listed if multiple segments are displayed at that position (the segments overlap).

3.1.2 Views

Colada displays data in various views. The user can choose between a number of predefined layouts, which may contain slice, 3D, chart, and table views.

The Layout Toolbar provides a drop-down menu of layouts useful for many types of studies. When Colada is exited normally, the selected layout is saved and restored next time the application is started.

3.1.3 Application Menu

- **File**: Functions for loading a previously saved scene or individual datasets of various types, and for downloading sample datasets from the internet. An option for saving scenes and data is also provided here. **Add Data** allows loading data from files. **Save** opens the “Save Data” window, which offers a variety of options for saving all data or selected datasets.
- **Edit**: Contains an option for showing Application Settings, which allows users to customize appearance and behavior of Colada, such as modules displayed in the toolbar, application font size, temporary directory location, location of additional Colada modules to include.

- **View**: Functions for showing/hiding additional windows and widgets, such as **Extensions Manager** for installing extensions from Colada app store, **Error Log** for checking if the application encountered any potential errors, **Python Console** for getting a Python console to interact with the loaded data or modules, **show/hide toolbars**, or **switch view layout**.

3.1.4 Toolbar

Toolbar provides quick access to commonly used functions. Individual toolbar panels can be shown/hidden using menu: View / Toolbars section.

Module Selection toolbar is used for selecting the currently active "module". The toolbar provides options for searching for module names (`Ctrl + f` or click on magnify glass icon) or selecting from a menu. **Module history dropdown button** shows the list of recently used modules. **Arrow buttons** can be used for going back to/returning from previously used module.

Favorite modules toolbar contains a list of most frequently used modules. The list can be customized using menu: Edit / Application settings / Modules / Favorite Modules. drag&drop modules from the Modules list to the Favorite Modules list to add a module.

3.1.5 Status bar

This panel may display application status, such as current operation in progress. Clicking the little **X** icons displays the Error Log window.

3.1.6 Data trees

`Data trees` are dedicated to work with geological data. Clicking on checkbox makes this data visible in `Data` module and thus can be viewed on the views. Right click on items invokes pop-up menu with allowed actions on selected items. Right click on tree header also invokes pop-up menu with different actions available.

3.1.7 Scene filter

Scene filter may be used to filter (hide) nodes whoose attribute's value doesn't match the value in the input line. The same can be done by setting `Domain` in the combo box. Node attributes can be viewed in `Data` module.

3.2 Mouse & Keyboard Shortcuts

3.2.1 Generic shortcuts

3.2.2 Slice views

The following shortcuts are available when a slice view is active. To activate a view, click inside the view: if you do not want to change anything in the view, just activate it then do `right-click` without moving the mouse. Note that simply hovering over the mouse over a slice view will not activate the view.

3.2.3 3D views

The following shortcuts are available when a 3D view is active. To activate a view, click inside the view: if you do not want to change anything in the view, just activate it then do `right-click` without moving the mouse. Note that simply hovering over the mouse over a slice view will not activate the view.

> **Note:** Simulation if shortcuts not available on your device:
>
> - One-button mouse: instead of `right-click` do `Ctrl` + `click`
> - Trackpad: instead of `right-click` do `two-finger click`

3.2.4 Python console

The following shortcuts are available in the Python console.

Note that when code is pasted into an empty line then all the code in the clipboard is executed *at once*. If the current command line is not empty then the code from the clipboard is pasted into the console and executed *line by line*. When code is executed line by line, the behavior is different in that an empty input line immediately closes the current block, and output is printed after executing each line.

CHAPTER

FOUR

APPLICATION SETTINGS

4.1 Editing application settings

The application settings dialog allows users to customize application behavior.

After starting Colada, it can be accessed clicking in menu: `Edit` / `Application Settings`.

4.1.1 General

Default directories

Scene and each type of containers are stored in separate folders. This makes convenient to load/store data.

Application startup script

Application startup script can be used to launch any custom Python code when Colada application is started.

4.1.2 Modules

Skip loading

Select which type of modules to not load at startup. It is also possible to start Colada by temporarily disabling those modules (not saved in settings) by passing the arguments in the command line.

For example, this command will start Colada without any CLI loaded:

```
Colada.exe --disable-cli-modules
```

Note: To see list of possible arguments type: `./Colada --help` or `./Colada.exe --help | more` on Windows.

Show hidden modules

Some modules don't have a user interface, they are hidden from the module's list. For debugging purpose, it is possible to force their display

Temporary directory

Directory where modules can store their temporary outputs if needed.

Additional module paths

List of directories scanned at startup to load additional modules. Any CLI, Loadable or scripted modules located in these paths will be loaded.

It is also possible to start Colada by temporarily adding module paths (not saved in settings) by passing the arguments in the command line.

For example this command will start Colada trying to load CLIs found in the specified directory:

```
Colada.exe --additional-module-paths C:\path\to\lib\Colada-X.Y\cli-modules
```

Modules

List of modules loaded, ignored or failed to load in Colada. An unchecked checkbox indicates that module should not be loaded (ignored) next time Colada starts. A text color code is used to describe the state of each module:

- Black: module successfully loaded in Colada
- Gray: module not loaded because it has been ignored (unchecked)
- Red: module failed to load. There are multiple reasons why a module can fail to load.

Look at startup log outputs to have more information. If a module is not loaded in Colada (ignored or failed), all dependent modules won't be loaded. You can verify the dependencies of a module in the tooltip of the module.

You can filter the list of modules by untoggling in the advanced (>>) panel the "To Load", "To Ignore", "Loaded", "Ignored" and "Failed" buttons.

Home

Module that is shown when Colada starts up.

Favorites

List of modules that appear in the Favorites toolbar.

To add a module, drag&drop it from the *Modules* list above. Then use the advanced panel (>>) to reorganize/delete the modules within the toolbar.

4.1.3 Appearance

Style

The overall theme of Colada is controlled by the selected Style:

- Colada (default): it sets the style based on theme settings set by the operating system. For example, on Windows if dark mode is turned on for apps, then the `Dark Colada` style will be used upon launching Colada. Currently, automatic detection of dark mode is not available on Linux, therefore use needs to manually select `Dark Colada` style for a dark color scheme.
- Light Colada: application window background is bright, regardless of operating system settings.
- Dark Colada: application window background is dark, regardless of operating system settings.

4.1.4 Developer

Developer mode

Enable some features that faciliate developer work like control for reloading, testing and editing scripted modules as well as restarting the application.

4.1.5 Spatial reference

To set spatial reference find one in the table and using drag&drop technique place it to input widget. If one doesn't want to use any spatial reference then clear current one and turn-on the checkbox `Ignore coordinate transformation on failure`.

4.2 Information for Advanced Users

4.2.1 Settings file location

Settings are stored in *.ini files. If the settings file is found in application home directory (within organization name or domain subfolder) then that .ini file is used. This can be used for creating a portable application that contains all software and settings in a relocatable folder. Relative paths in settings files are resolved using the application home directory, and therefore are portable along with the application.

If .ini file is not found in the the application home directory then it is searched in user profile:

- Windows: `%USERPROFILE%\AppData\Roaming\NA-MIC\` (typically `C:\Users\<your_user_name>\AppData\Roaming\NA-MIC\`)
- Linux: `~/.config/NA-MIC/`
- Mac: `~/.config/www.na-mic.org/`

Deleting the *.ini files restores all the settings to default.

There are two types of settings: `user specific settings` and `user and revision specific settings`.

User specific settings

This file is named `Colada.ini` and it stores settings applying to *all versions* of Colada installed by the *current user*.

To display the exact location of this settings file, open a terminal and type:

```
./Colada --settings-path
```

On Windows:

```
Colada.exe --settings-path | more
```

or enter the following in the Python console:

```
slicer.app.slicerUserSettingsFilePath
```

User and revision specific settings

This file is named like `Colada-<REVISION>.ini` and it stores settings applying to a *specific revision* of Colada installed by the *current user*.

To display the exact location of this settings file, enter the following in the Python console:

```
slicer.app.slicerRevisionUserSettingsFilePath
```

4.2.2 Application startup file

Each time Colada starts, it will look up for a startup script file named .slicerrc.py. Content of this file is executed automatically at each startup of Colada.

The file is searched at multiple location and the first one that is found is used. Searched locations:

- Application home folder (`slicer.app.slicerHome`)
- Path defined in `SLICERRC` environment variable
- User profile folder (`~/.slicerrc.py`)

You can find the path to the startup script in Colada by opening in the menu: Edit / Application Settings. ''Application startup script'' path is shown in the ''General'' section (or running `getSlicerRCFileName()` command in Colada Python console).

4.2.3 Runtime environment variables

The following environment variables can be set before the application is started to fine-tune its behavior:

- `PYTHONNOUSERSITE`: if it is set to `1` then import of user site packages is disabled. For example, this will prevent Colada to reuse packages downloaded/built by Anaconda.
- `QT_SCALE_FACTOR`: see Qt documentation. For example, font size can be reduced by running `set QT_SCALE_FACTOR=0.5` in the command console and then starting Colada in that console.
- `QT_ENABLE_HIGHDPI_SCALING`: see Qt documentation
- `QT_SCALE_FACTOR_ROUNDING_POLICY`: see Qt documentation
- `QTWEBENGINE_REMOTE_DEBUGGING`: port number for Qt webengine remote debugger. Default value is `1337`.

- SLICER_OPENGL_PROFILE: Requested OpenGL profile. Valid values are no (no profile), core (core profile), and compatibility (compatibility profile). Default value is compatibility on Windows systems.
- SLICER_BACKGROUND_THREAD_PRIORITY: Set priority for background processing tasks. On Linux, it may affect the entire process priority. An integer value is expected, default = 20 on Linux and macOS, and -1 on Windows.
- SLICERRC: Custom application startup file path. Contains a full path to a Python script. By default it is ~/.slicerrc.py (where ~ is the user profile a.k.a user home folder).
- SLICER_EXTENSIONS_MANAGER_SERVER_URL: URL of the extensions manager backend with the /api path. Default value is retrieved from the settings using the key Extensions/ServerUrl.
- SLICER_EXTENSIONS_MANAGER_FRONTEND_SERVER_URL: URL of the extension manager frontend displaying the web page. Default value is retrieved from the settings using the key Extensions/FrontendServerUrl.
- SLICER_EXTENSIONS_MANAGER_SERVER_API: Supported value is Girder_v1. Default value is hard-coded to Girder_v1.

4.2.4 Qt built-in command-line options

Colada application accepts standard Qt command-line arguments that specify how Qt interacts with the windowing system.

Examples of options:

- -qwindowgeometry geometry, specifies window geometry for the main window using the X11-syntax. For example: -qwindowgeometry 100x100+50+50.
- -display hostname:screen_number, switches displays on X11 and overrides the DISPLAY environment variable.
- -platform windows:dpiawareness=[0|1|2], sets the DPI awareness on Windows.
- -widgetcount, prints debug message at the end about number of widgets left undestroyed and maximum number of widgets existed at the same time.
- -reverse, sets the application's layout direction to Qt::RightToLeft.

To learn about the supported options:

- https://doc.qt.io/qt-5/qapplication.html#QApplication
- https://doc.qt.io/qt-5/qguiapplication.html#supported-command-line-options

Note: Since the Colada launcher is itself a Qt application and the Qt built-in command-line options are expected to **only** be passed to the launched application ColadaApp-real and not the Colada launcher, the list of arguments to filter is specified in the Main.cpp found in the commontk/AppLauncher project.

CHAPTER

FIVE

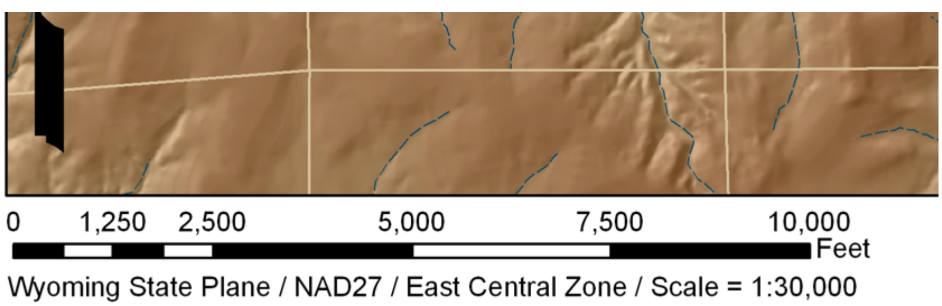
TUTORIALS

The following tutorials are available:

5.1 IO and Visualization

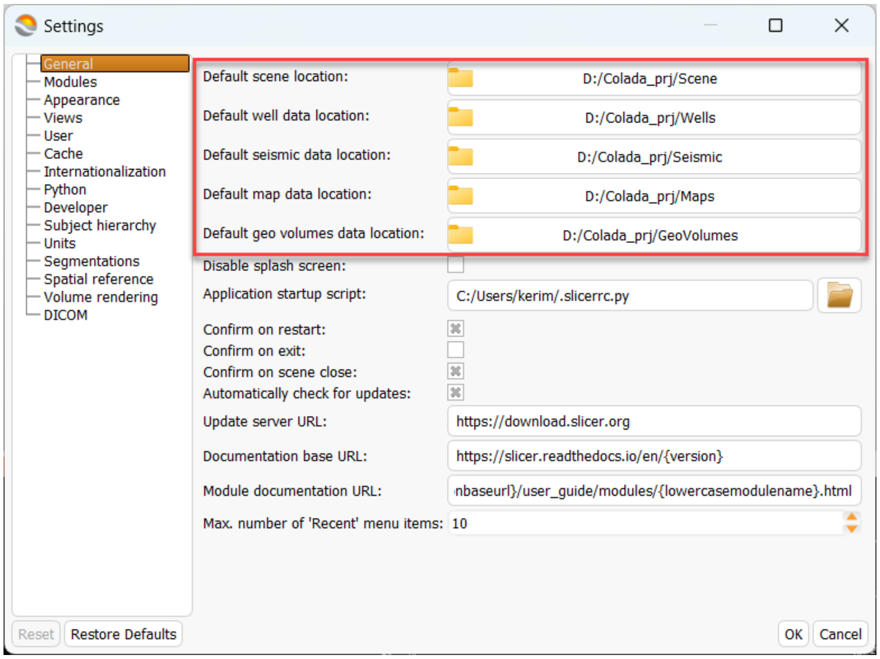
Start by dowloading Teapot dome 3D open source dataset.

Next we need to find information about spatial refernce used in downloaded data. For that go to `DataSets/GIS/CD files` and open `NPR3_BASEMAP.jpg` that shows the survey on the relief map. In the lower left corner there is a spatial reference and units:

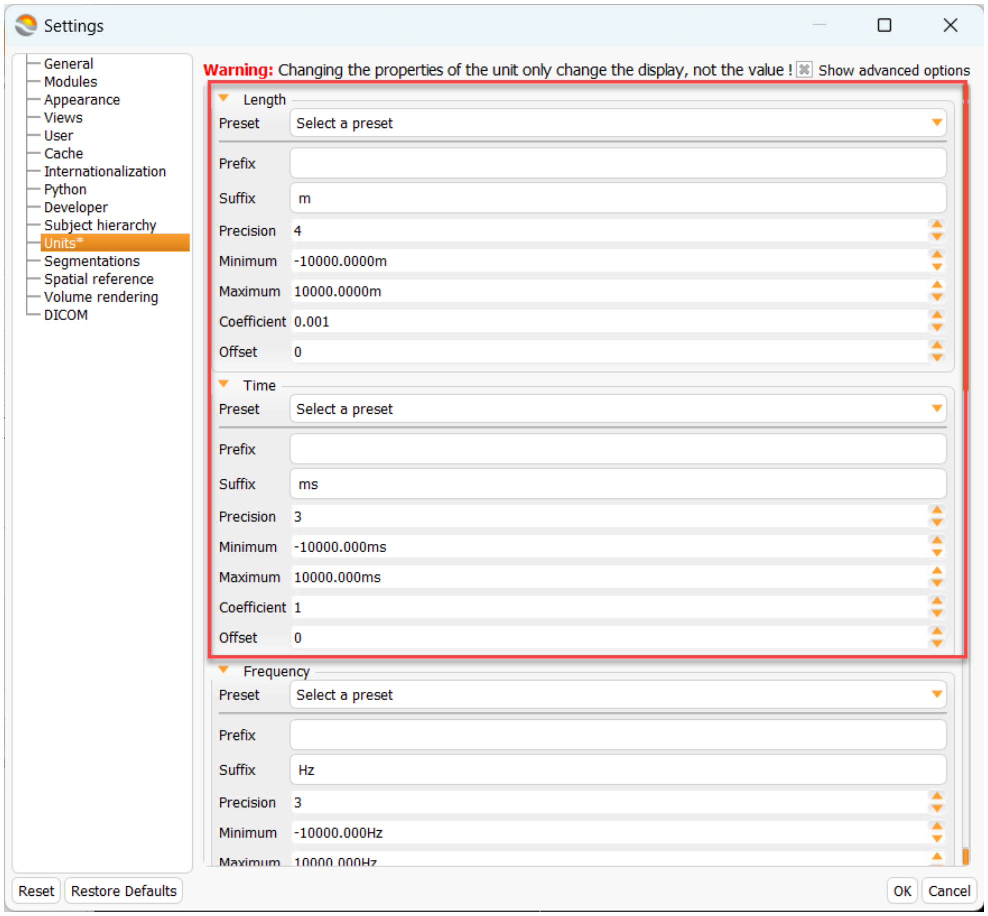


5.1.1 Application settings

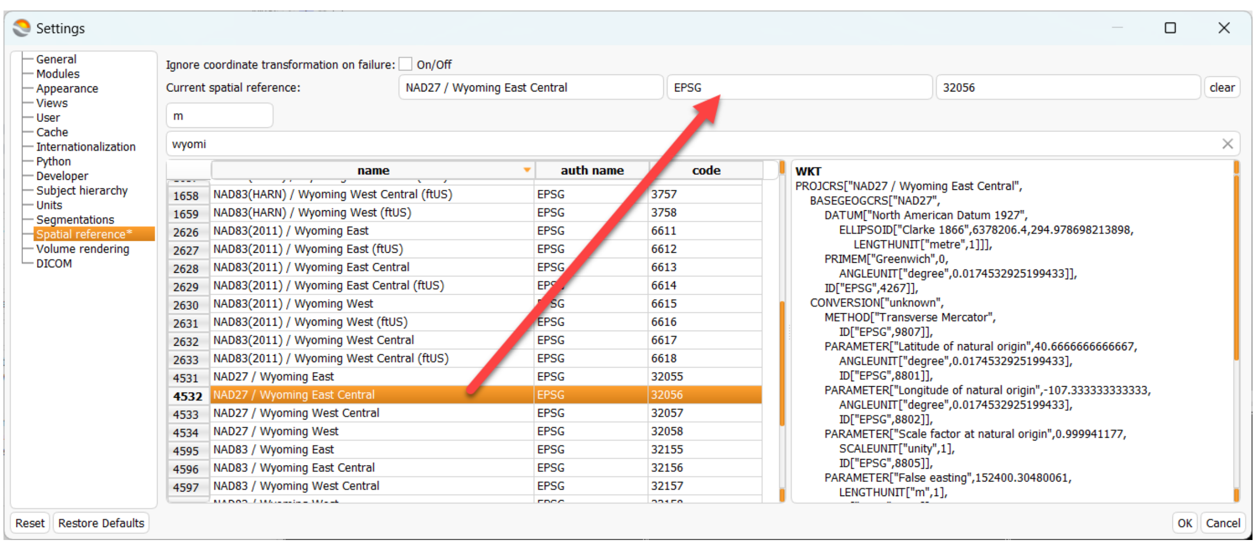
In Colada application settings (`Toolbar menu->Edit->Application Settings`) `General` panel set default directories for the data.



In the `Units` panel set preffered length and temporal units. In my case those are `meter` and `millisecond`.



Then go to the spatial reference panel and find the appropriate one. It seems that `NAD27 / Wyoming East Central (EPSG:32056)` is the right choice. Set it using drag&drop.



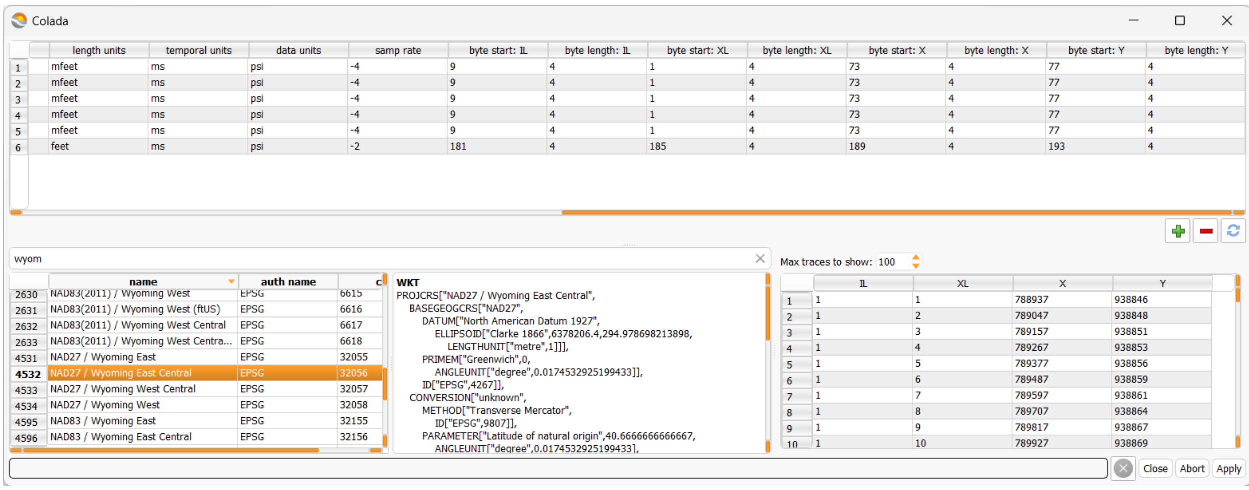
5.1.2 Read SEGY

SEGY is represented by 3D STACK (`Seismic/CD files/3D_Seismic/filt_mig.sgy`) and bunch of 2D STACK (`DataSets/Seismic/CD files/2D_Seismic`) datasets.

In Colada SEGY STACK may be read as *Geo Volume* or as *Seismic* object. We will read it as Geo Volume first and then as Seismic.

Read SEGY as Geo Volume

To open Geo Volume SEGY reader right-click on Geo Volume tree `Import->SEGY STACK`. Add SEGY files from `DataSets/Seismic/CD files/2D_Seismic/NormalizedMigrated_segy/` and `DataSets/Seismic/CD files/3D_Seismic/` directories. Before reading SEGY don't forget to check trace headers. For *Teapot Dome* the correct header offsets for INLINE, XLINE and X, Y are shown in the picture.



Warning: Geo Volume is a regular grid that can be represented by origin, orientation and spacings along each axis.

SEGY stack data may not strictly follow this. In this case XY plane will be broken and in practice the user must be confident that the SEGY is represented as a cube or flat with regular spacings.

There are few things that should be explained.

First of all each geo-object has attribute `Domain`. Possible values are: `TWT` (Two Way Traveltime), `OWT` (One Way Traveltime), `TVD` (True Vertical Depth), `TVDSS` (True Vertical Depth Sub-Sea).

CRS (Coordinate Seference System or Spatial Reference System) highly desirable for every geo-object.

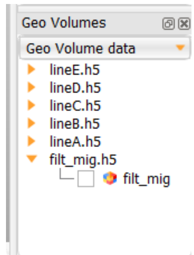
`Length units`, `Temporal units` necessary for all geo-objects. Uusually they are needed to define coordinates.

`Data units` necessary for some geo-objects like Map if it is in `TWT/OWT` domain. Or for Geo Volume if it is going to be used in wave-modeling as velocity model for example. `Data units` defines units of the data, for example for a Map object it defines units of surface values. Or for Geo Volume it defines units of each scalar value of data (pressure for example or particle displacement).

To be confident that your units are convertable one may use the web service.

Geo Volume visualization

After data is read it should appear in the Geo Volume tree. As Geo Volume may be pretty big and not all the time the user wants to load oll data to the memory, after clicking on the checkbox the module will be switched to `GeoVolumes` where one can select a subset of the data to load.

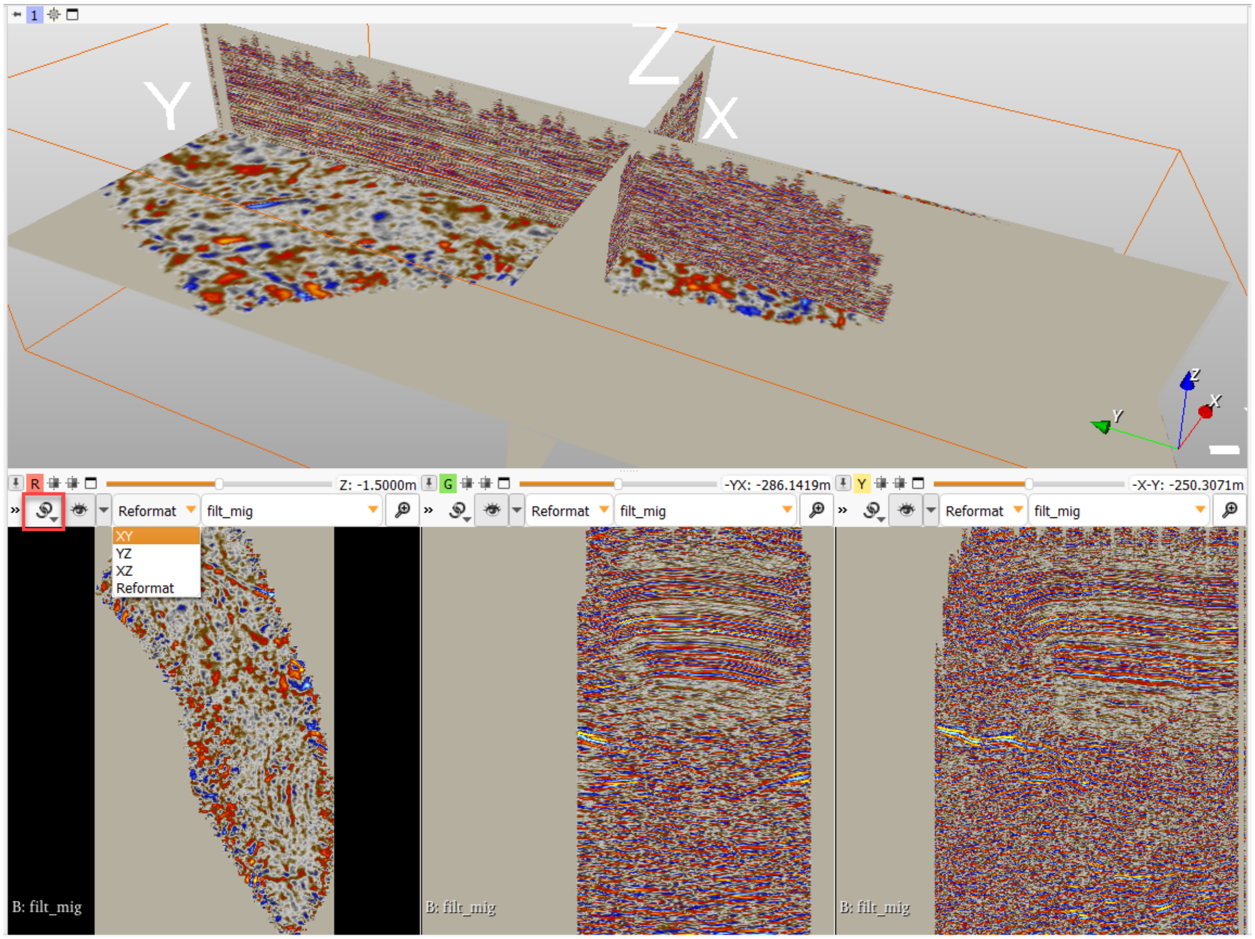


When the data is loaded new item will appear in *Subject Hierarchy* window. To visualize Geo Volume one may click on *eye* picture like shown in the picture.

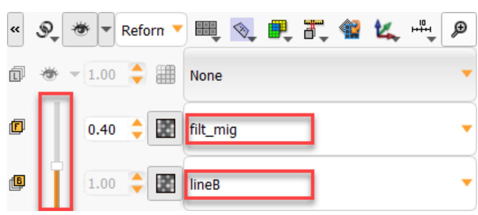
Geo Volume should be visualized.

By default *Slice Views* are synchronized. One can disable taht by clicking on the *chain* button.

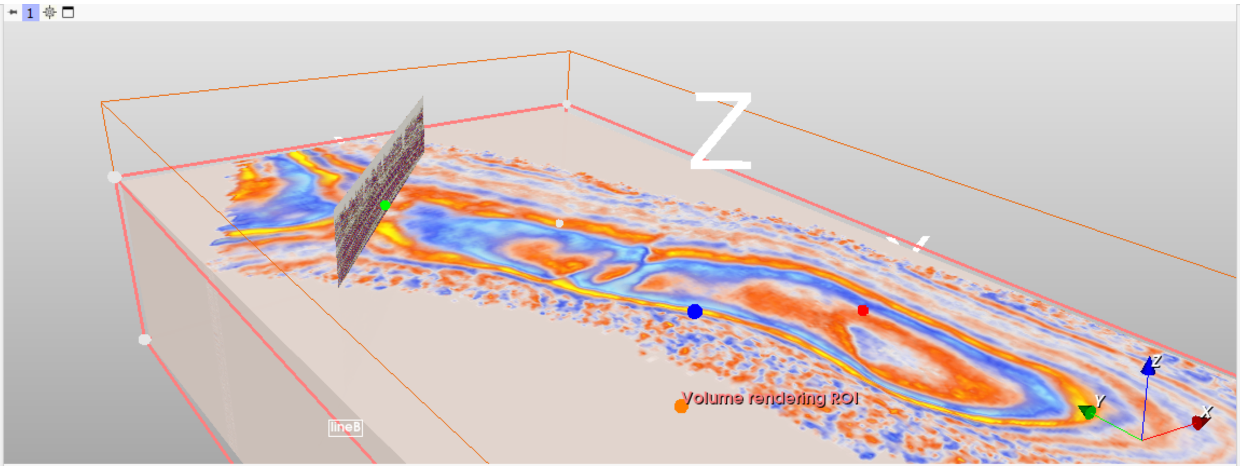
To see axes in Slice View one should change the preset to one of the following: `XY`, `YZ`, `XZ`.



Further one can visualize two volumes one upon another. To do that there is a place for *background* and *foreground* volumes. Place 2D as background and 3D cuba as foreground. Then using slicer change the opacty to see how these two objects relates.



Another way to visualize the data is to drag&drop the selected object to the scene. In this case an object will be rendered as Volume.



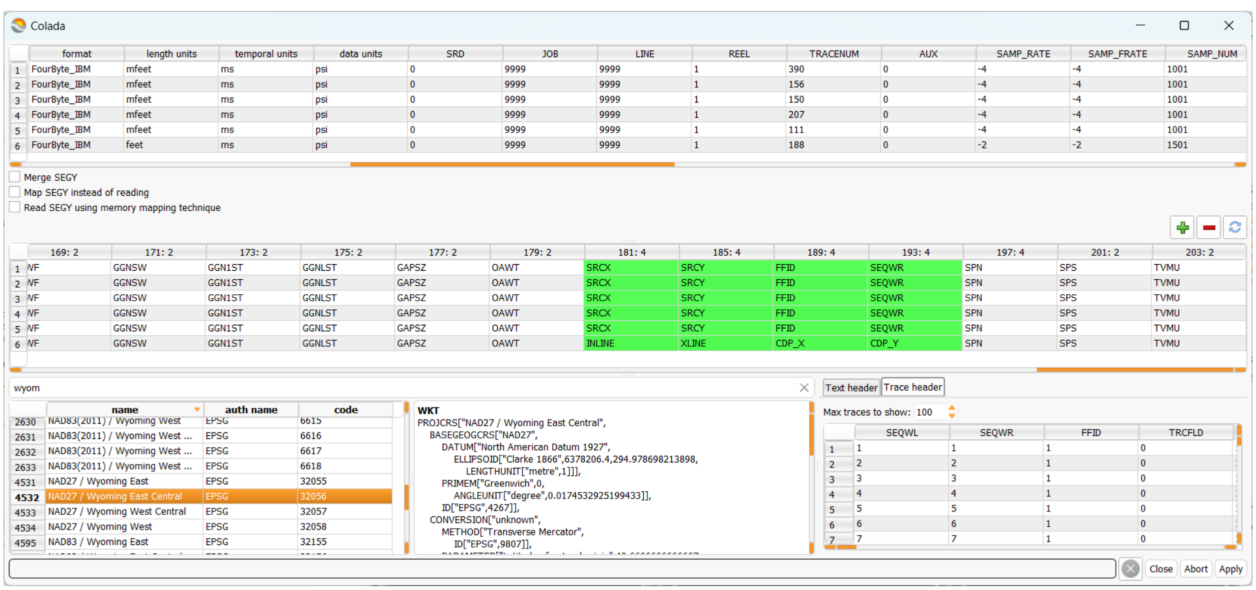
Rendered Volume is controlled in `Volume Rendering` module. `Volume` module is for usual representation using Slices. Check these modules to see how you can customize the visualization.

Read SEGY as Seismic

To read SEGY as Seismic right click on *Seismic* treeview to invoke menu `Import->SEGY`. Add the same files as we have read with Geo Volume reader.

Carefully set all the parameters. it is important to set survey type (2D/3D), data type (stack/prestack), domain, units ,sampling rate.

Note: Colada uses `Z` values decreases downwards. That means when reading usual SEGY file as Seismic one need to set negative sampling rate.



Provided SEGY files have mixed trace headers. `INLINE/XLINE` are either missing or switched with `CDP_X/CDP_Y` (for 3D). `CDP_X/CDP_Y` are switched with `SRCX/SRCY` (for 2D).

To fix that there is table in the middle of the window. The point is to set the correct trace header names for all trace header offsets. If you do this correctly then all modified items in the table will be colored by green color. Be sure there is no red colored boxes.

Seismic visualization

When working with Seismic it usually requires some kind of a sorting. As Colada provides means to work with STACK as well as with STACK seismic the user should be familiar with seismic sorting.

Colada doesn't resort data itself but it rather writes trace indices for every primary key value (`pKey`). For example if we want to get `INLINE->XLINE` then we need to prepare sorting for `INLINE` as it is a primary key. The sorting is efficient when the number of unique values a lot less of the amount of traces. Thus sorting for `CDP/SP/INLINE/XLINE` etc are pretty effective.

So click on the checkbox in the Seismic Tree and *Seismic* module appear. Expand sorting tab and add `INLINE` sorting.

Then `X/Y` coord headers as `CDP_X/CDP_Y` and load it.

Seismic may be loaded as:

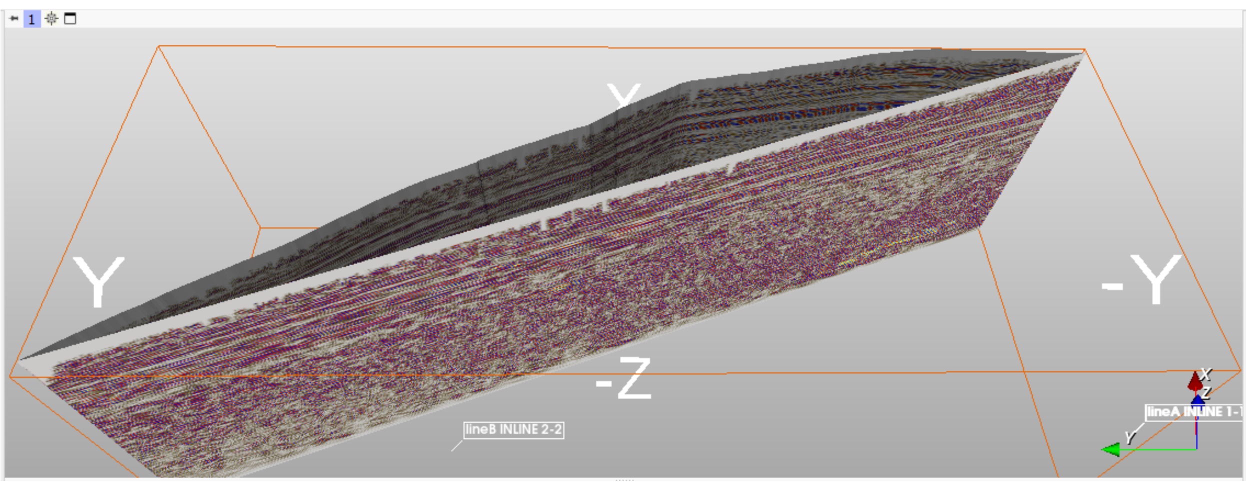
1) Volume (rectangular grid, 3D)
2) Volumetric mesh (unstructured grid, 3D)
3) Surface (2D)

Volume uses interpolation if the original data has random XY coordinates.

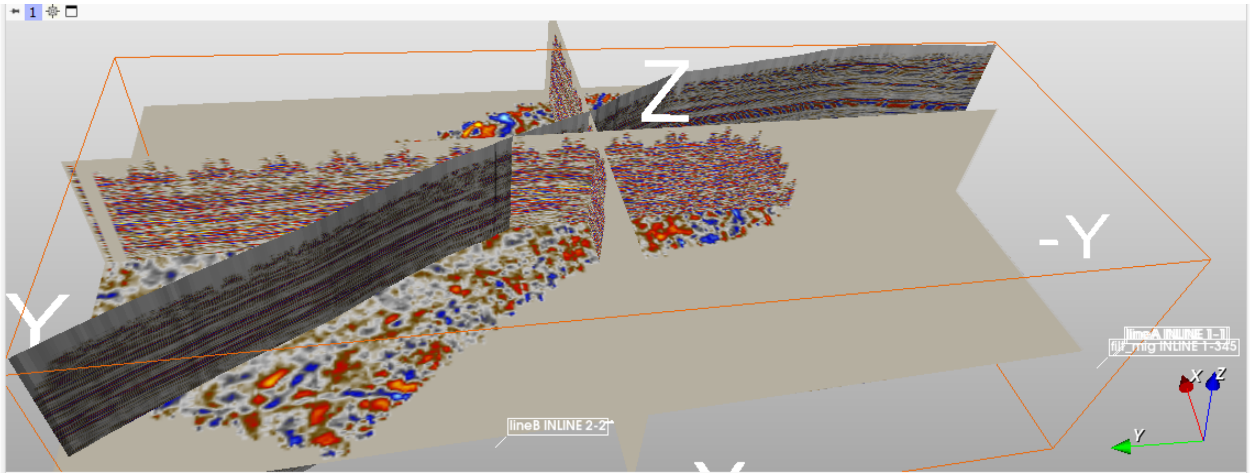
Volumetric mesh use data as is but it requires about four times more RAM tan Volume.

Surface is for 2D selected data.

For example we can load the same 2D seismic `lineA` as Volume and then as Surface and see the difference.



In the same way we can visualize 3D seismic as volume.



Volume visualization is controlled by `Volumes` and `Volume Rendering` modules.

Surface and Volumetric mesh is controlled by `Models` module.

All the loaded data is displayed in `Data` module.

Try to play with it.

5.1.3 Read Well data

In `DataSets/Well Log/CD Files` there are `DirectionalSurveys_020910.xlsx`, `TeapotDomeWellHeaders02-09-10.xlsx`, `TeapotDomeFormationLogTops.xlsx`. This is not a standardized file format. To read it we will use *Python*. We can read it using `Pandas` package for example. Or preinstall desired package using `slicer.util.pip_install("my_package")` and use it. But as I already did it using `csv` we will export Excel to `csv` *comma* delimited file and then process it.

Note: To run the Python code one either need to copy-paste it to the Interpreter or save it as script and `Ctrl` + `g`. Or from command line `./Colada --python-script <path-to-file>`

Read Well Heads

This snippet reads well heads from *comma* delimited `TeapotDomeWellHeaders02-09-10.csv` file (don't forget to change `readfile` path at the *bottom*).

```
from PythonQt import *
from qColadaAppPythonQt import *
from h5geopy import h5geo
import numpy as np
import csv


def readTeapotWellHeads(readfile: str, savefile: str):
  def is_float(element) -> bool:
    try:
      float(element)
      return True
```

(continues on next page)

(continued from previous page)

```
    except ValueError:
      return False
  # set parameters for the newly created well
  p_w = h5geo.H5WellParam()
  p_w.spatialReference = h5geo.sr.getAuthName() + ':' + h5geo.sr.getAuthCode()
  p_w.lengthUnits = 'feet'
  p_w.temporalUnits = 'ms'
  p_w.angularUnits = 'degree'
  # set parameters for the newly created devcurve
  p_d = h5geo.H5DevCurveParam()
  p_d.spatialReference = h5geo.sr.getAuthName() + ':' + h5geo.sr.getAuthCode()
  p_d.lengthUnits = 'feet'
  p_d.temporalUnits = 'ms'
  p_d.angularUnits = 'degree'
  p_d.chunkSize = 1 # we set only signle row to curve. Chunking affects on the datasize
  wellname = ''
  kb = 0
  head_xy = np.zeros(2, order='F')
  # create hdf5 well container
  h5wellCnt = h5geo.createWellContainerByName(savefile, h5geo.CreationType.CREATE_OR_
↪OVERWRITE)
  if not h5wellCnt:
    raise ValueError(f"Unable to create or overwrite Well Container: {savefile}")
  with open(readfile,) as csvfile:
    QtGui.QApplication.setOverrideCursor(QtCore.Qt.BusyCursor)
    reader = csv.reader(csvfile, dialect='excel')
    for line in reader:
      if reader.line_num < 3:
        continue
      if len(line) < 11 or not is_float(line[4]) or not is_float(line[5]) or not is_
↪float(line[9]):
        continue
      if 'kb' not in line[10].lower():
        continue
      p_w.uwi = line[0][5:10]  # info https://en.wikipedia.org/wiki/API_well_number
      wellname = line[3].replace('/', '') # symbol `/` creates new group in hdf5
      head_xy[0] = float(line[5])
      head_xy[1] = float(line[4])
      kb = float(line[9])
      # create well (skip if fail)
      h5well = h5wellCnt.createWell(wellname, p_w, h5geo.CreationType.CREATE_OR_
↪OVERWRITE)
      if not h5well:
        continue
      h5well.setHeadCoord(head_xy)
      h5well.setKB(kb)
      # create devcurve (skip if fail)
      h5devCurve = h5well.createDevCurve('vertical trajectory', p_d, h5geo.CreationType.
↪CREATE)
      if not h5devCurve or not is_float(line[8]):
        continue
      md = np.zeros(2)
```

(continues on next page)

(continued from previous page)

```
      md[1] = float(line[8])  # assuming that well is vertical then MD = TVD
      azim = np.zeros(2)
      incl = np.zeros(2)
      h5devCurve.writeMD(np.asfortranarray(md, dtype=float))
      h5devCurve.writeAZIM(np.asfortranarray(azim, dtype=float))
      h5devCurve.writeINCL(np.asfortranarray(incl, dtype=float))
      h5devCurve.updateTvdDxDy()
      if not h5well.openActiveDevCurve():
        h5devCurve.setActive()
    QtGui.QApplication.restoreOverrideCursor()
    print(f"Well headers file read to: {savefile}")

# ===============================
# CHANGE THE PATH TO THE CSV FILE
# ===============================
readfile = r'E:\\Teapot Dome\\DataSets\\Well Log\\CD Files\\TeapotDomeWellHeaders02-09-
↪10.csv'
savefile = Util.defaultWellDir() + "/wells.h5"


readTeapotWellHeads(readfile,savefile)
```

Read Deviations

Implemented deviation reader is based on format like those exported by *Petrel*. In this example the format is Excel. Export it to csv and run:

```
from PythonQt import *
from qColadaAppPythonQt import *
from h5geopy import h5geo
import numpy as np


def readTeapotDeviations(readfile: str, savefile: str):
  def is_float(element) -> bool:
    try:
      float(element)
      return True
    except ValueError:
      return False
  p_w = h5geo.H5WellParam()
  p_w.spatialReference = h5geo.sr.getAuthName() + ':' + h5geo.sr.getAuthCode()
  p_w.lengthUnits = 'feet'
  p_w.temporalUnits = 'ms'
  p_w.angularUnits = 'degree'
  p_d = h5geo.H5DevCurveParam()
  p_d.spatialReference = h5geo.sr.getAuthName() + ':' + h5geo.sr.getAuthCode()
  p_d.lengthUnits = 'feet'
  p_d.temporalUnits = 'ms'
  p_d.angularUnits = 'degree'
  p_d.chunkSize = 10
```

(continues on next page)

(continued from previous page)

```
h5wellCnt = h5geo.createWellContainerByName(savefile, h5geo.CreationType.OPEN_OR_
↪CREATE)
if not h5wellCnt:
  raise ValueError(f"Unable to open or create Well Container: {savefile}")
h5well = None
h5devCurve = None
m = np.zeros([0,3])
need_write = False
dev_curve_name = None
with open(readfile) as file:
  QtGui.QApplication.setOverrideCursor(QtCore.Qt.BusyCursor)
  for line in file:
    strlist = line.split()
    if len(strlist) > 1 and 'well' in strlist[0].lower():
      m = np.zeros([0,3])
      need_write = True
    elif (len(strlist) > 3 and is_float(strlist[1]) and
        is_float(strlist[2]) and is_float(strlist[3])):
      p_w.uwi = strlist[0][5:10]
      dev_curve_name = strlist[0][10:12]  # info https://en.wikipedia.org/wiki/API_
↪well_number
      m = np.append(m, np.zeros([1,3]), axis=0)
      m[-1,0] = float(strlist[1]) # MD
      m[-1,1] = float(strlist[3]) # AZIMUTH
      m[-1,2] = float(strlist[2]) # INCLINATION
    elif len(strlist) < 1 and m.shape[0] > 0 and need_write:
      if not dev_curve_name or not p_w.uwi:
        continue
      # create well (skip if fail)
      h5well = h5wellCnt.openWellByUWI(p_w.uwi)
      if not h5well:
        continue
      h5devCurve = h5well.createDevCurve(dev_curve_name, p_d, h5geo.CreationType.OPEN_
↪OR_CREATE)
      if not h5devCurve:
        continue
      h5devCurve.setActive()
      h5devCurve.writeMD(np.asfortranarray(m[:,0], dtype=float))
      h5devCurve.writeAZIM(np.asfortranarray(m[:,1], dtype=float))
      h5devCurve.writeINCL(np.asfortranarray(m[:,2], dtype=float))
      h5devCurve.updateTvdDxDy()
      need_write = False
  if need_write and dev_curve_name and p_w.uwi:
    h5well = h5wellCnt.openWellByUWI(p_w.uwi)
    if h5well:
      # create devcurve (skip if fail)
      h5devCurve = h5well.createDevCurve(dev_curve_name, p_d, h5geo.CreationType.OPEN_
↪OR_CREATE)
      if h5devCurve:
        h5devCurve.setActive()
        h5well.setUWI(p_w.uwi)
        h5devCurve.writeMD(np.asfortranarray(m[:,0], dtype=float))
```

(continues on next page)

(continued from previous page)

```
          h5devCurve.writeAZIM(np.asfortranarray(m[:,1], dtype=float))
          h5devCurve.writeINCL(np.asfortranarray(m[:,2], dtype=float))
          h5devCurve.updateTvdDxDy()
    QtGui.QApplication.restoreOverrideCursor()
    print(f"Dev file read to: {savefile}")

# ===============================
# CHANGE THE PATH TO THE CSV FILE
# ===============================
readfile = r'E:\\Teapot Dome\\DataSets\\Well Log\\CD Files\\DirectionalSurveys_020910.csv
↪'
savefile = Util.defaultWellDir() + "/wells.h5"


readTeapotDeviations(readfile, savefile)
```

Read Well Tops

Once again export `TeapotDomeFormationLogTops.xlsx` to `csv` and run:

```
from PythonQt import *
from qColadaAppPythonQt import *
from h5geopy import h5geo
import numpy as np
import csv


def readTeapotWellTops(readfile: str, savefile: str):
  p_wt = h5geo.H5WellTopsParam()
  p_wt.nPoints = 1
  p_wt.chunkSize = 5
  p_wt.domain = h5geo.Domain.TVD
  p_wt.spatialReference = h5geo.sr.getAuthName() + ':' + h5geo.sr.getAuthCode()
  p_wt.lengthUnits = 'feet'
  data = {}
  wellname_old = ''
  wellname_new = ''
  h5wellCnt = h5geo.createWellContainerByName(savefile, h5geo.CreationType.OPEN)
  if not h5wellCnt:
    raise ValueError(f"Unable to open Well Container: {savefile}")
  with open(readfile,) as csvfile:
    QtGui.QApplication.setOverrideCursor(QtCore.Qt.BusyCursor)
    reader = csv.reader(csvfile, dialect='excel')
    for line in reader:
      if reader.line_num < 2 or len(line) != 4:
        continue
      if reader.line_num == 2:
        wellname_old = line[1]
      else:
        wellname_old = wellname_new
      wellname_new = line[1]
```

(continues on next page)

(continued from previous page)

```
        if wellname_old != wellname_new:
          h5well = h5wellCnt.openWell(wellname_old)
          if not h5well:
            continue
          h5welltops = h5well.createWellTops(p_wt, h5geo.CreationType.OPEN_OR_CREATE)
          if not h5welltops:
            continue
          p_wt.nPoints = len(data)
          p_wt.chunkSize = len(data)
          h5welltops.writeData(data, 'feet')
        topsname = line[2]
        md = float(line[3])
        data[topsname] = md
      QtGui.QApplication.restoreOverrideCursor()
      print(f"Well Tops file read to: {savefile}")

# ===============================
# CHANGE THE PATH TO THE CSV FILE
# ===============================
readfile = r'E:\\Teapot Dome\\DataSets\\Well Log\\CD Files\\TeapotDomeFormationLogTops.
↪csv'
savefile = Util.defaultWellDir() + "/wells.h5"


readTeapotWellTops(readfile,savefile)
```

Read LAS

Fortunately LAS standardized format and thus there is graphical interface for reading it. Right click on Well Tree and `Import->LAS`. From `DataSets/Well Log/CD Files/LAS_log_files/Deeper_LAS_files/` directory add LAS files to the reader.

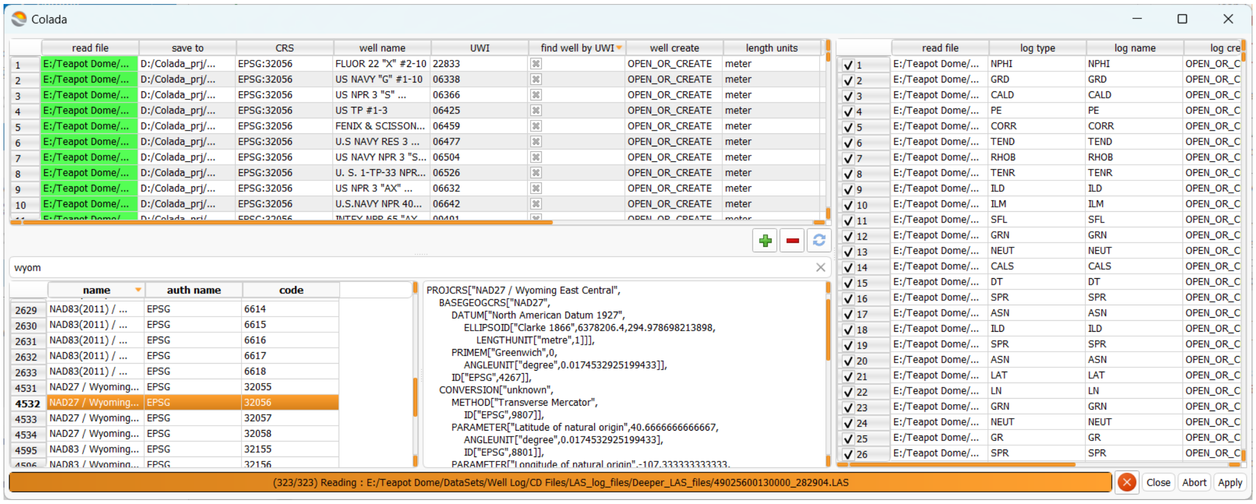
Wait, this will take a while.

Change `save to` for the container used for reading well heads, well tops and deviations (`/some/path/wells.h5`). Actively use copy&paste for that.

Probably it is worth to check on the box `find well by UWI` as well names and not always correct. For that select the whole column and check on the box: this will cause all the column to be checked.

In the right table one can exclude/modify curves.

Reading LAS takes some time.

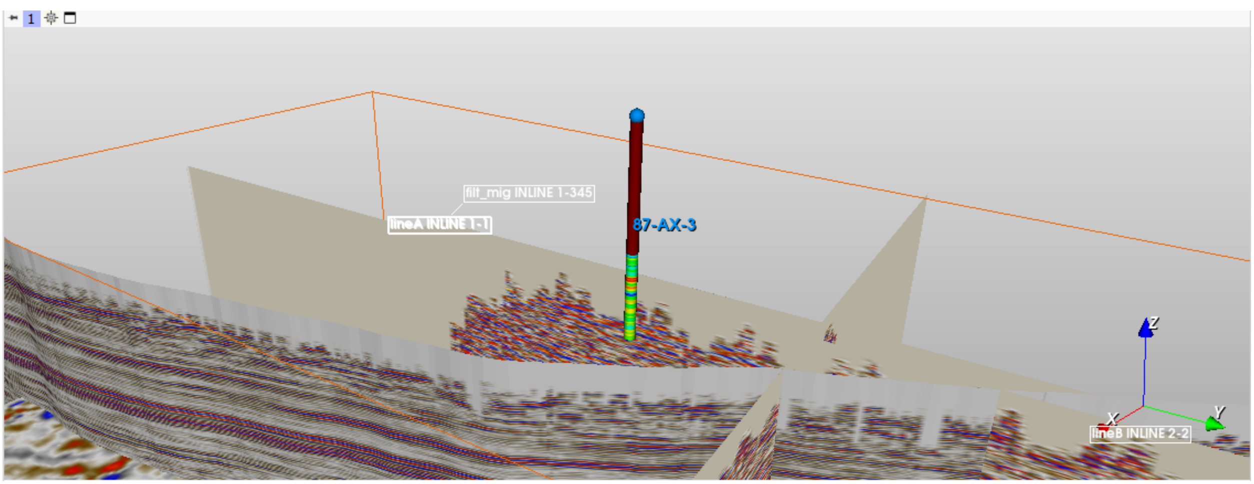


Well Visualization

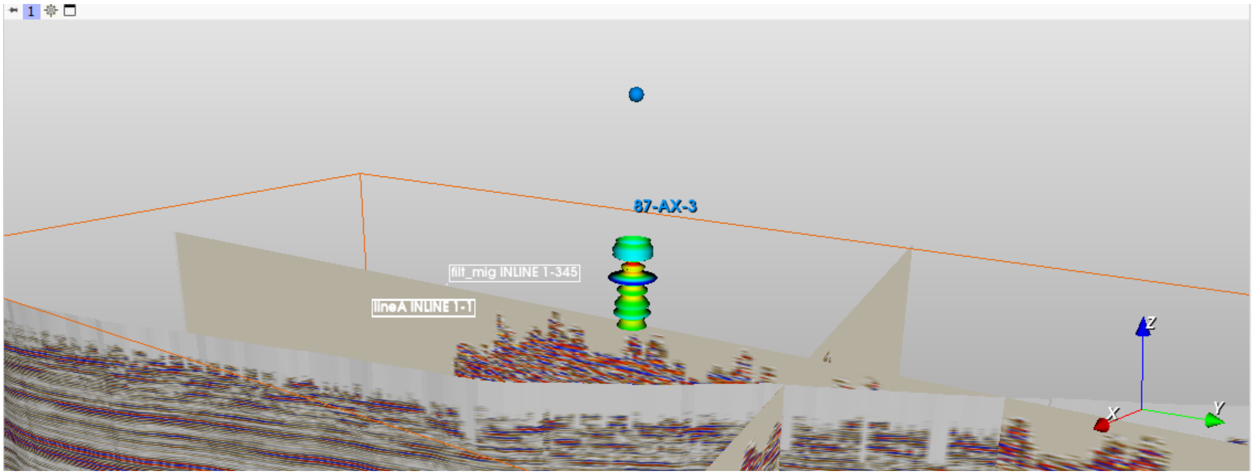
After it is finished go to the Well treeview select some wells and check them. If you are lucky wells appear in the scene. If not then choose `87-AX-3` it contains both trajectory and logs. Go to the *Wells* module and select loaded well. A number of deviations and logs should appear in the tables.

Load trajectory using 200 resampled points. Then load `CALD/CALD` log.

The scene now should look like in the picture:



Then check on `Variant Thickness` and adjust radius and smooth.



Further customizations are done in `Markups` module. Try to play with it.

5.2 Wave Modeling: simple 2D model

Before starting to work with wave modeling/migration/inversion we must have at least two things:

1) velocity model
2) geometry

Velocity model is a Geo Volume object. It may be read from SEGY or be prepared using Python/Julia interface.

Geometry is either *H5Seis* object or SEGY files. It includes at least Source/Receiver positions (XYZ coordinates). Geometry may be generated using *Seismic* module under *Geometry* section. It is pretty straightforward.

In this tutorial we will use Python interface to build model and generate geometry.

5.2.1 2D velocity model building

Let's start from setting initial data to build model with two horizontal layers:

```
from h5geopy import h5geo
import numpy as np
import scipy as sp
import scipy.ndimage
import colada
import slicer
import os

# to be able to open hdf5 container in HDFVIEW
os.environ["HDF5_USE_FILE_LOCKING"] = "FALSE"

# setting output directories
out_vol_dir = colada.Util().defaultGeoVolumesDir() +'/'
out_seis_dir = colada.Util().defaultSeisDir() +'/'
out_volcnt_name = "model_2D.h5"
out_vol_name = "model_2D"
out_vol_name_smooth = "model_2D_smooth"
```

(continues on next page)

(continued from previous page)

```
out_geomcnt_name = "geom_2D.h5"
out_geom_name = "geom_2D"

SPATIAL_REFERENCE = h5geo.sr.getAuthName() + ":" + h5geo.sr.getAuthCode()
LENGTH_UNITS = "m"
TEMPORAL_UNITS = "ms"
ANGULAR_UNITS = "degree"
DATA_UNITS = "km/s"
ORIENTATION = 0.0

# model params (x,z)
o = (0,0)
d = (12.5,12.5)
n = (300,100)
n1 = int(round(n[1]/3)) # first reflector depth
n2 = int(round(2*n[1]/3)) # second reflector depth

# velocities for the layers
v1 = 1.5
v2 = 2
v3 = 2.5

# is used to smooth velocity model
smooth_radius = 20
```

Most of this should be pretty understandable The most important here is model parameters like size `n` of the model, origin `o`, spacings `d`. `v1,v2,v3` define first, second and third layer velocities respectively. `smooth_radius` defines the number of points per axis to apply filter to the model to get smoothed velocity model.

Second step is to create velocity model represented by `numpy` array. In our case the model is very simple:

```
x = np.arange(o[0],o[0]+d[0]*(n[0]-1),d[0])
z = np.arange(o[1],o[1]+d[1]*(n[1]-1),d[1])

data = np.zeros(n[::-1]) # reverse `n` so the `data.shape = [nz,nx]`
data[:n1,:] = v1
data[n1:n2,:] = v2
data[n2:,:] = v3
data_smooth = sp.ndimage.uniform_filter(data, smooth_radius)

# flip Z axis as h5geo stores volumes in low-to-high direction (origin is lower left␣
↪corner)
# also make the order layout to Fortran (h5geo works with column oriented arrays)
data = np.asfortranarray(np.flip(data,0), dtype=np.float32)
data_smooth = np.asfortranarray(np.flip(data_smooth,0), dtype=np.float32)
```

The most important here is to understand which direction correspond to wich axis. If we work with 2D model then numpy array should be of shape `[nz,nx]`. `data` is our model (`numpy` array). `data_smooth` is smoothed model. Here is the trick: Colada uses origin as lower left corner of the model. Thus we need to flip `z` axis.

Also it worth to notice that `h5geo` works only with `1D` and `2D` Fortran memory ordered arrays and the writable data usually stored as `float32`.

Note: Pay attension that we will alway use Fortran memory layout when working with `h5geo`. And don't forget to cast data to 1D or 2D array of `float32`.

The last step is to create `H5Seis` object within seismic container and write the generated model:

```
p = h5geo.H5VolParam()
p.spatialReference = SPATIAL_REFERENCE
p.lengthUnits = LENGTH_UNITS
p.temporalUnits = TEMPORAL_UNITS
p.angularUnits = ANGULAR_UNITS
p.dataUnits = DATA_UNITS
p.domain = h5geo.Domain.TVDSS
p.orientation = ORIENTATION
p.X0 = o[0]
p.Y0 = 0.0
p.Z0 = o[1]-d[1]*(n[1]-1)
p.dX = d[0]
p.dY = 1.0
p.dZ = d[1]
p.nX = n[0]
p.nY = 1
p.nZ = n[1]
p.xChunkSize = 64
p.yChunkSize = 1
p.zChunkSize = 64
p.compression_level = 6

# create Geo Volume container
volcnt = h5geo.createVolContainerByName(
      out_vol_dir+out_volcnt_name,
      h5geo.CreationType.OPEN_OR_CREATE)
if not volcnt:
  raise RuntimeError(f"Unable to create Volume container: {out_volcnt_name}")

# create Geo Volume object (H5Vol)
vol = volcnt.createVol(out_vol_name, p, h5geo.CreationType.CREATE_OR_OVERWRITE)
if not vol:
  raise RuntimeError(f"Unable to create Geo Volume: {out_vol_name}")

# write the data to Geo Volume
status = vol.writeData(data.ravel(), 0,0,0,p.nX,p.nY,p.nZ, DATA_UNITS)
if not status:
  raise RuntimeError(f"Unable to write data to Geo Volume: {out_vol_name}")

# create Geo Volume for smotthed model
volsmooth = volcnt.createVol(out_vol_name_smooth, p, h5geo.CreationType.CREATE_OR_
↪OVERWRITE)
if not volsmooth:
  raise RuntimeError(f"Unable to create Geo Volume: {out_vol_name_smooth}")

# write smoothed data to Geo Volume
status = volsmooth.writeData(data_smooth.ravel(), 0,0,0,p.nX,p.nY,p.nZ, DATA_UNITS)
```

(continues on next page)

(continued from previous page)

```
if not status:
  raise RuntimeError(f"Unable to write data to Geo Volume: {out_vol_name_smooth}")

# add geo volume container to treeview
slicer.util.mainWindow().emitH5FileToBeAdded(volcnt.getH5File().getFileName())

# don't forget to close h5geo (hdf5) objects
del vol
del volsmooth
del volcnt
```

Warning: *h5geo* objects hold reference to *h5gt* object wich is *hdf5* wrapper. It is **extremely important** to close these objects right after you've done working with them (delete variable is one of the possible ways). If file is opened and `HDF5_USE_FILE_LOCKING=TRUE` then this file will be unavailable for other processes. If `HDF5_USE_FILE_LOCKING=FALSE` then you are risking to break the file if its content is modified from another process.

The variable `p` is used to create new `H5Vol` object. Almost all its fields are self descridable.

`X0,Y0,Z0` define origin (in legth units). `dX,dY,dZ` are spacings aling `X,Y,Z` axes respectively. `nX,nY,nZ` are dimensions. `orientation` is orientation around `Z` axis.

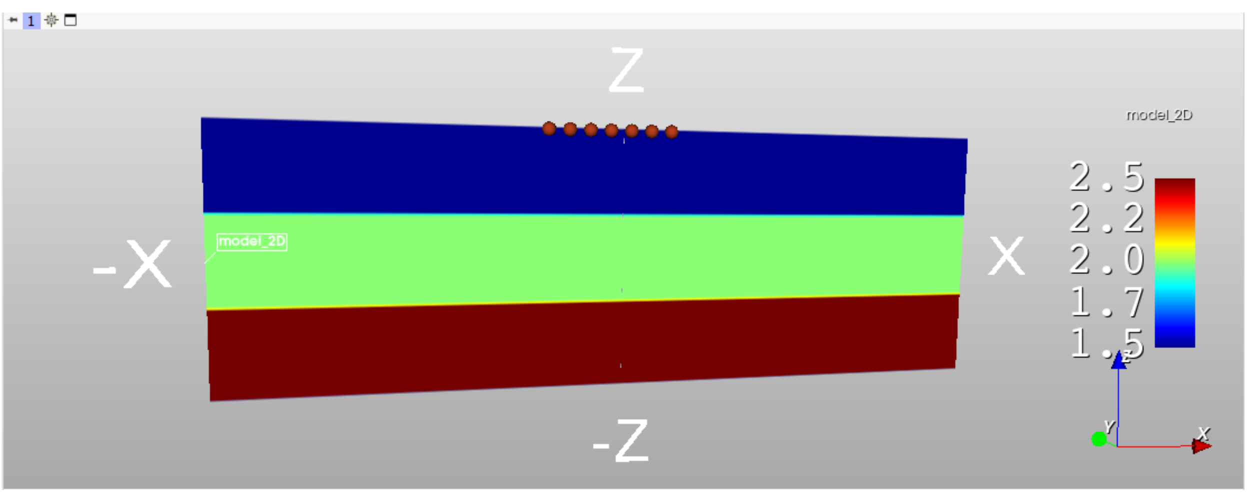
Chunking is important for IO perfomance. In this case we are trying to imitate `ZGY` format using chunking equal to 64. `compression_level` is used to compress the data.

Note: Geo Volume must not contain zeroed or negative spacings (i.e. it must have `dX,dY,dZ > 0`). Also `nX,nY,nZ` must be > `0`.

Velocity model visualization

Generated model should be visible in Geo Volume treeview.

Load Geo Volume. The picture should be similar to this one:



5.2.2 Geometry for 2D model

There is no need to write the algorithm for calculating geometry. It is already done in `h5geo`.

So we start by setting the initial data for geometry:

```
# timings
nSamp = 800
sampRate = -2.0

# geometry
src_x0 = 1500
src_dx = 100
src_nx = 8
src_y0 = 0
src_dy = 0
src_ny = 1
src_z = o[1]

rec_x0 = 0
rec_dx = 50
rec_nx = 61
rec_y0 = 0
rec_dy = 0
rec_ny = 1
rec_z = o[1]
moveRec = True
```

Then we simpy fill `H5SeisParam` fields to create new `H5Seis` object within seismic container and call the function `generatePRESTKGeometry` to generate PRESTACK geometry:

```
# h5geo parameters to create new H5Seis object
p = h5geo.H5SeisParam()
p.spatialReference = SPATIAL_REFERENCE
p.lengthUnits = LENGTH_UNITS
p.temporalUnits = TEMPORAL_UNITS
p.angularUnits = ANGULAR_UNITS
p.dataUnits = "psi"
p.domain = h5geo.Domain.TWT
p.dataType = h5geo.SeisDataType.PRESTACK
p.surveyType = h5geo.SurveyType.THREE_D
p.nTrc = 100
p.nSamp = nSamp
p.srd = 0
p.trcChunk = 100
p.stdChunk = 100

# create Seismic container
geomcnt = h5geo.createSeisContainerByName(
      out_seis_dir + out_geomcnt_name,
      h5geo.CreationType.OPEN_OR_CREATE)
if not geomcnt:
  raise RuntimeError(f"Unable to create Seis container: {out_geomcnt_name}")
```

(continues on next page)

(continued from previous page)

```
# create H5Seis object
geom = geomcnt.createSeis(out_geom_name, p, h5geo.CreationType.CREATE_OR_OVERWRITE)
if not geom:
  raise RuntimeError(f"Unable to create Seis: {out_geom_name}")

# setting sampling rate
geom.setSampRate(sampRate)

# generate geometry
status = geom.generatePRESTKGeometry(
    src_x0, src_dx, src_nx,
    src_y0, src_dy, src_ny,
    src_z,
    rec_x0, rec_dx, rec_nx,
    rec_y0, rec_dy, rec_ny,
    rec_z,
    ORIENTATION,
    moveRec)

if not status:
  raise RuntimeError(f"Unable to generate geometry: {out_geom_name}")

# add geometry container to treeview
slicer.util.mainWindow().emitH5FileToBeAdded(geomcnt.getH5File().getFileName())

# don't forget to close h5geo (hdf5) objects
del geom
del geomcnt
```

The model has length 3737.5m. Source geometry contains 8 sources at positions from 1500 to 2200m with 100m step. Receivers are moved along with sources (`moveRec = True`), that means for the first source receivers are spread from 0m to 3000m with 50m step. For the second source receivers are spread from 100m to 3100m and so on.

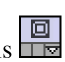
Geometry check

Geometry container should be added to the Seismic treeview.

First thing to do is to add sorting. As we have PRESTACK geometry then we can add `SRCX` sorting. For that go to the `Seismic` module, set active seismic at the top (one may use drag&drop from treeview), open *Sorting* section and double click on `SRCX` header. This will cause `SRCX` to appear in the right list. Then click `Add Sorting`.

After that `SRCX` sorted data may be loaded either as any type of mesh or loaded to table. We will load it to table and plot.

Open *Load to Table* section, click on combobox *Select a Table->Create new Table*. Most widgets become enabled. In the `Sorted trace headers` section set `SRCX` as *primary key* (PKey). Then click on *plus* button to add secondary key (SKey) and set it to `GRPX` and click *Load sorted trace headers*.

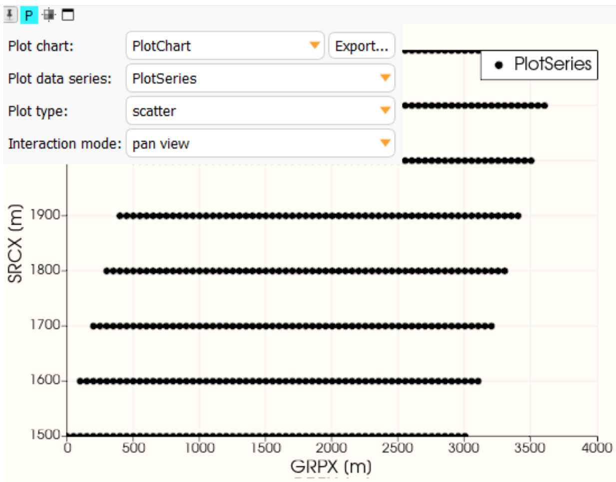
Change layout by clicking the button on the toolbar similar to this  and choose one containing *plot*.

Go to the *Plots* module click `Create new PlotChart`. This simply creates new plot. To add data to it click `Create new PlotSeries`. Then click on `Series` tab. Set the parameters:

- Plot type *Scatter*

- Input table: *Table*
- X Axis column: *GRPX*
- Y Axis column: *SRCX*
- Marker style: *square*
- Line style: *None*

After some manipulations one can achieve something like that:



Note: Loaded trace headers are also available in *Tables* module. But be aware that tables with n*1000 rows work slow and may greatly reduce the perfomance.

5.2.3 Forward Modeling

Go to the `Wave Modeling` module.

Physical Parameters:

- Velocity: previously generated `model_2D.h5`

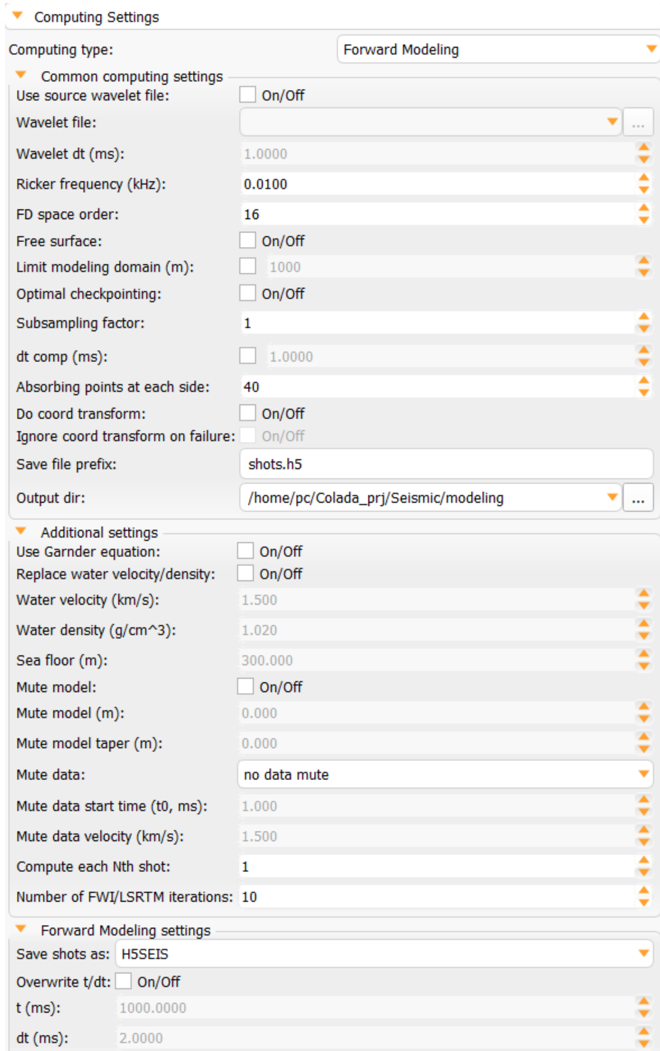
Field Records (Geometry):

- File format: H5SEIS
- Geometry: previously generated `geom_2D.h5`
- PKey (Primary Key): SP
- Source x: SRCX
- Receiver x: GRPX

Computing Settings:

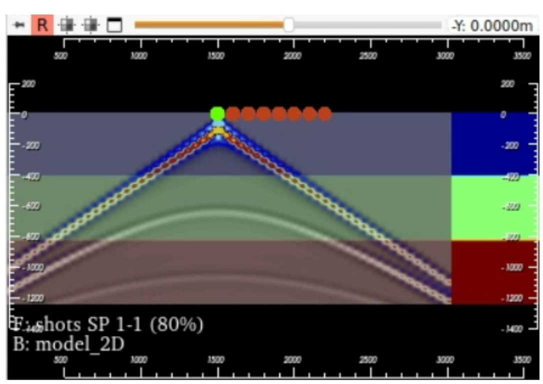
- Computing Type: Forward Modeling
- Set *Save file prefix* (`shots.h5`) and *Output dir*
- Other settings may be changed by your choice

After settings are set something similar should be resulted:



Note: Muting is not supposed to be used with *Forward Modeling*.

After calculations are done we can visualize model overlayed by computed shots (first shot for example):



5.2.4 Reverse Time Migration (RTM)

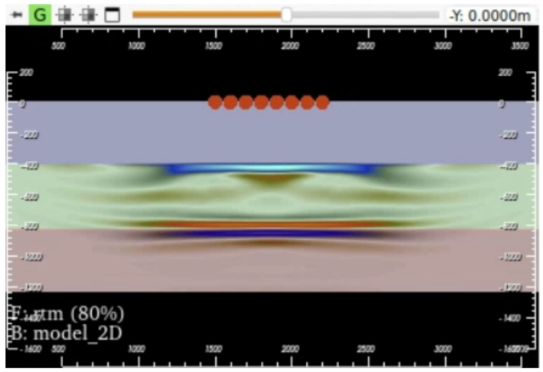
To compute RTM first of all we need to set previously computed shots as geometry.

Then in *Computing Settings* set *Computing type: RTM*.

Set *Save file prefix* as *rtm.h5* and *Output dir* to Geo Volumes.

To achieve good results we should apply muting. Basically we should mute water layer (412.5m) and turning waves (Data mute). We will use model muting 412.5m without taper. In *Mute data* set *mute turning*. *Mute t0* to 150ms (approximate wavelet length) and *Mute data velocity* to 1.5km/s (water velocity).

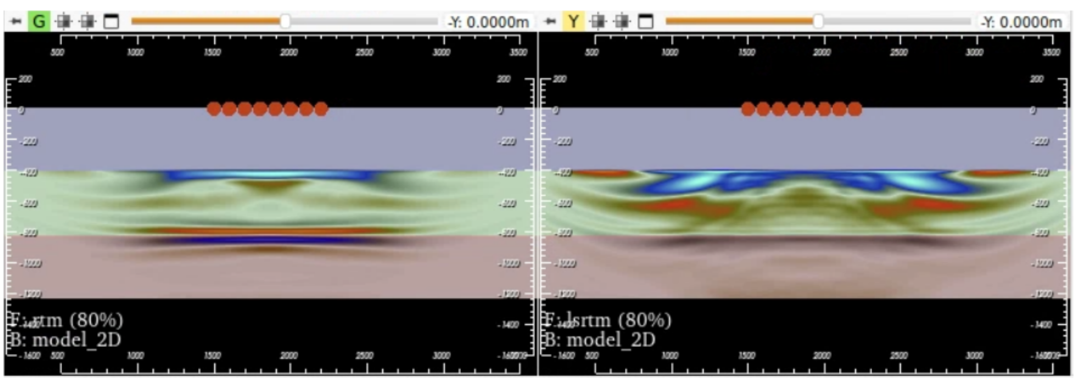
2D velocity model overlayed with RTM is shown below:



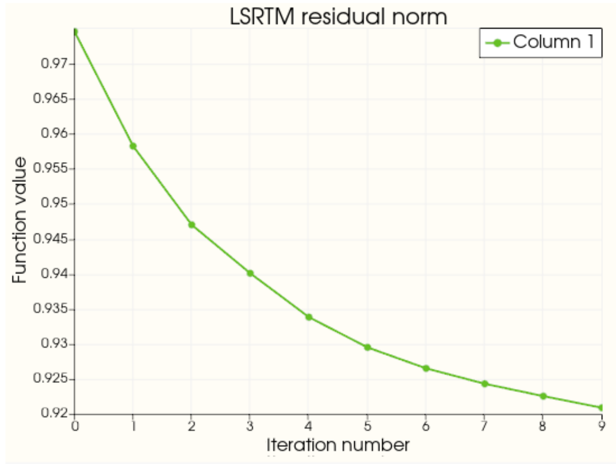
Note: In practice field data must contain only reflected waves (our data have both direct and reflected waves). Also in this example source step is too big (500m). That is why the result is not clean.

5.2.5 Least-Squares Reverse Time Migration (LSRTM)

To run LSRTM select true velocity model as *Input*. *Geometry* same generated shots. In *Computing Settings* set *Save file prefix* to *lsrtm.h5* and *Output dir* to Geo Volumes. Use same mutings that were used in conventional RTM and run.



Residual norm and other convergence attribute may be found in created *lsrtm.h5* container (use HDFVIEW):

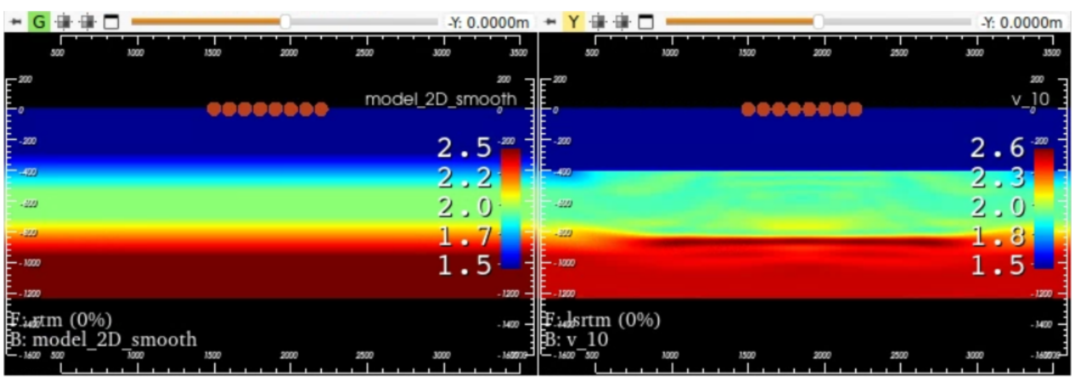


5.2.6 Full Waveform Inversion (FWI)

As we know the goal of FWI is to make starting velocity model more precise. Thus we need to set smoothed velocity model *model_2D_smooth* as input. The geometry is the same shots that were used for RTM. And in *Computing Settings* we need to set *Computing type: FWI*. *Save file prefix* as *fwi.h5* and *Output dir* to Geo Volumes. Model muting is 412.5m without taper. We will not use *Data muting* but in practice FWI works with turnings waves. And **important thing** is to turn on *Replace water velocity/density*. This will replace velocity (and density if present) at each iteration.

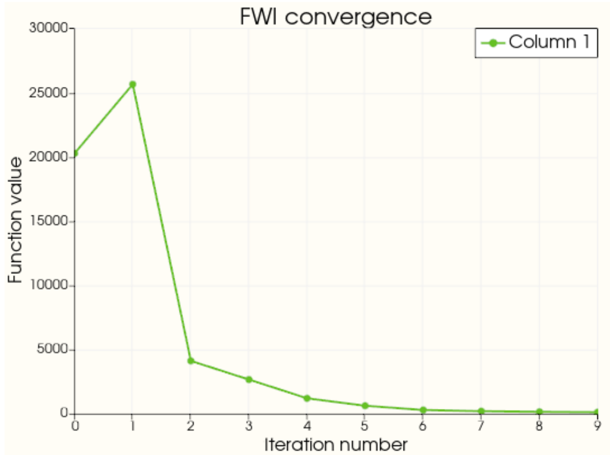
In *FWI Settings* set min/max velocities to 1000 and 3500 respectively.

After passing 10 iteration we may get result similar to shown below:



At left is the smoothed starting model and at right - result of FWI 10th iteration.

To see the convergence we can copy values from *fhistory* dataset (open with HDFVIEW for example) and copy them to new table in *Tables* module. Then plot it.



Congratulations! We hope you enjoyed this tutorial.

5.3 Wave Modeling: simple 3D model

This tutorial is identical to *Simple Wave Modeling 2D* but is aimed at 3D modeling. We suppose you are familiar with the previous tutorials.

5.3.1 3D velocity model building

Start model building from setting initial parameters:

```
from h5geopy import h5geo
import numpy as np
import scipy as sp
import scipy.ndimage
import colada
import slicer
import os

# to be able to open hdf5 container in HDFVIEW
os.environ["HDF5_USE_FILE_LOCKING"] = "FALSE"

# setting output directories
out_vol_dir = colada.Util().defaultGeoVolumesDir() +'/'
out_seis_dir = colada.Util().defaultSeisDir() +'/'
out_volcnt_name = "model_3D.h5"
out_vol_name = "model_3D"
out_vol_name_smooth = "model_3D_smooth"
out_geomcnt_name = "geom_3D.h5"
out_geom_name = "geom_3D"

SPATIAL_REFERENCE = h5geo.sr.getAuthName() + ":" + h5geo.sr.getAuthCode()
LENGTH_UNITS = "m"
TEMPORAL_UNITS = "ms"
ANGULAR_UNITS = "degree"
DATA_UNITS = "km/s"
```

(continues on next page)

(continued from previous page)

```
ORIENTATION = 0.0

# model params (x,z)
o = (0,0,0)
d = (12.5,12.5,12.5)
n = (300,200,100)
n1 = int(round(n[2]/3)) # first reflector depth
n2 = int(round(2*n[2]/3)) # second reflector depth

# velocities for the layers
v1 = 1.5
v2 = 2
v3 = 2.5

# is used to smooth velocity model
smooth_radius = 20
```

The model size is [nx,ny,nz]=[300,200,100] points. Distance between two nearest points is equal at each direction and equal to 12.5m. Thus model length in each direction [x,y,z]=[3737.5m,2487.5m,1237.5m]. Velocities of three layers: 1.5, 2, 2.5km/s.

The following code builds velocity model:

```
x = np.arange(o[0],o[0]+d[0]*(n[0]-1),d[0])
y = np.arange(o[1],o[1]+d[1]*(n[1]-1),d[1])
z = np.arange(o[2],o[2]+d[2]*(n[2]-1),d[2])

data = np.zeros(n[::-1]) # reverse `n` so the `data.shape = [nz,ny,nx]`
data[:n1,:,:] = v1
data[n1:n2,:,:] = v2
data[n2:,:,:] = v3
data_smooth = sp.ndimage.uniform_filter(data, smooth_radius)

# flip Z axis as h5geo stores volumes in low-to-high direction (origin is lower left␣
↪corner)
# also make the order layout to Fortran (h5geo works with column oriented arrays)
data = np.asfortranarray(np.flip(data,0), dtype=np.float32)
data_smooth = np.asfortranarray(np.flip(data_smooth,0), dtype=np.float32)
```

Now we need to write previously built velocity model to `H5Vol` object (Geo Volume):

```
p = h5geo.H5VolParam()
p.spatialReference = SPATIAL_REFERENCE
p.lengthUnits = LENGTH_UNITS
p.temporalUnits = TEMPORAL_UNITS
p.angularUnits = ANGULAR_UNITS
p.dataUnits = DATA_UNITS
p.domain = h5geo.Domain.TVDSS
p.orientation = ORIENTATION
p.X0 = o[0]
p.Y0 = o[1]
p.Z0 = o[2]-d[2]*(n[2]-1)
p.dX = d[0]
```

(continues on next page)

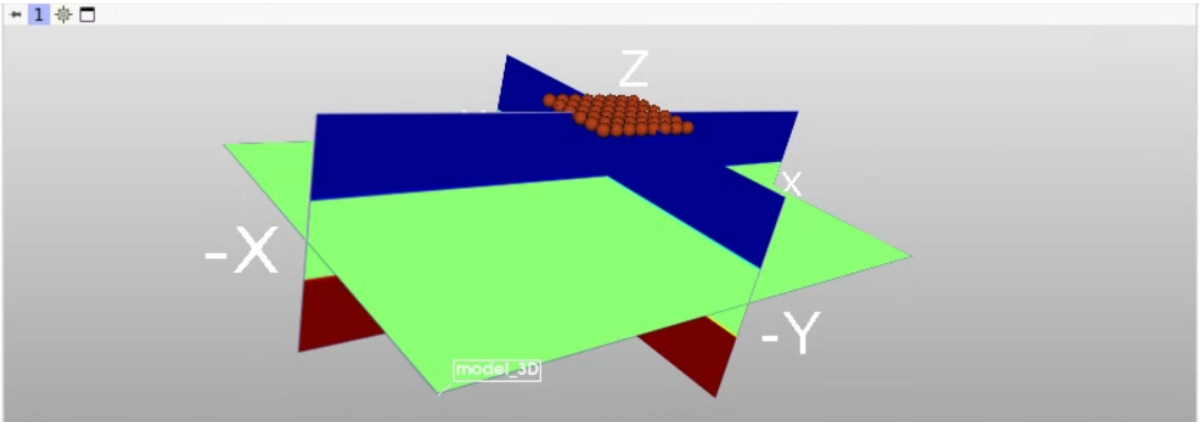
(continued from previous page)

```
p.dY = d[1]
p.dZ = d[2]
p.nX = n[0]
p.nY = n[1]
p.nZ = n[2]
p.xChunkSize = 64
p.yChunkSize = 64
p.zChunkSize = 64
p.compression_level = 6

volcnt = h5geo.createVolContainerByName(
      out_vol_dir+out_volcnt_name,
      h5geo.CreationType.OPEN_OR_CREATE)
if not volcnt:
  raise RuntimeError(f"Unable to create Volume container: {out_volcnt_name}")

vol = volcnt.createVol(out_vol_name, p, h5geo.CreationType.CREATE_OR_OVERWRITE)
if not vol:
  raise RuntimeError(f"Unable to create Geo Volume: {out_vol_name}")

status = vol.writeData(data.ravel(), 0,0,0,p.nX,p.nY,p.nZ, DATA_UNITS)
if not status:
  raise RuntimeError(f"Unable to write data to Geo Volume: {out_vol_name}")

volsmooth = volcnt.createVol(out_vol_name_smooth, p, h5geo.CreationType.CREATE_OR_
↪OVERWRITE)
if not volsmooth:
  raise RuntimeError(f"Unable to create Geo Volume: {out_vol_name_smooth}")

status = volsmooth.writeData(data_smooth.ravel(), 0,0,0,p.nX,p.nY,p.nZ, DATA_UNITS)
if not status:
  raise RuntimeError(f"Unable to write data to Geo Volume: {out_vol_name_smooth}")

# add geo volume container to treeview
slicer.util.mainWindow().emitH5FileToBeAdded(volcnt.getH5File().getFileName())

# don't forget to close h5geo (hdf5) objects
del vol
del volsmooth
del volcnt
```

Warning: *h5geo* objects hold reference to *h5gt* object wich is *hdf5* wrapper. It is **extremely important** to close these objects right after you've done working with them (delete variable is one of the possible ways). If file is opened and `HDF5_USE_FILE_LOCKING=TRUE` then this file will be unavailable for other processes. If `HDF5_USE_FILE_LOCKING=FALSE` then you are risking to break the file if its content is modified from another process.

Velocity model visualization

Picture below is prepared 3D velocity model visualized in Colada:



5.3.2 Geometry for 3D model

Start by setting initial data for 3D geometry:

```
# timings
nSamp = 800
sampRate = -2.0

# geometry
src_x0 = 1500
src_dx = 100
src_nx = 8
src_y0 = 1000
src_dy = 100
src_ny = 6
src_z = o[2]

rec_x0 = 1000
rec_dx = 50
rec_nx = 21
rec_y0 = 500
rec_dy = 50
rec_ny = 21
rec_z = o[2]
moveRec = True
```

Fill H5SeisParam to create new H5Seis object and generate PRESTACK geometry:

```
# h5geo parameters to create new H5Seis object
p = h5geo.H5SeisParam()
p.spatialReference = SPATIAL_REFERENCE
p.lengthUnits = LENGTH_UNITS
p.temporalUnits = TEMPORAL_UNITS
p.angularUnits = ANGULAR_UNITS
p.dataUnits = "psi"
```

(continues on next page)

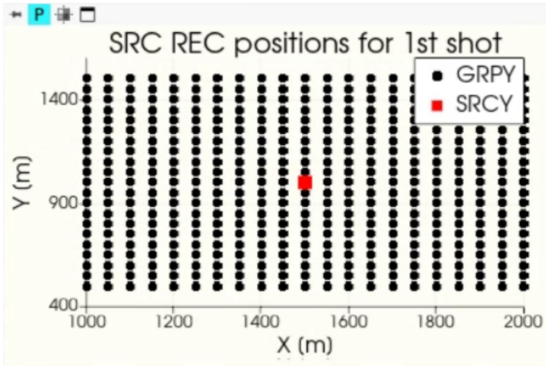
(continued from previous page)

```
p.domain = h5geo.Domain.TWT
p.dataType = h5geo.SeisDataType.PRESTACK
p.surveyType = h5geo.SurveyType.THREE_D
p.nTrc = 100
p.nSamp = nSamp
p.srd = 0
p.trcChunk = 1000
p.stdChunk = 100

# create Seismic container
geomcnt = h5geo.createSeisContainerByName(
      out_seis_dir + out_geomcnt_name,
      h5geo.CreationType.OPEN_OR_CREATE)
if not geomcnt:
  raise RuntimeError(f"Unable to create Seis container: {out_geomcnt_name}")

# create H5Seis object
geom = geomcnt.createSeis(out_geom_name, p, h5geo.CreationType.CREATE_OR_OVERWRITE)
if not geom:
  raise RuntimeError(f"Unable to create Seis: {out_geom_name}")

# setting sampling rate
geom.setSampRate(sampRate)

# generate geometry
status = geom.generatePRESTKGeometry(
    src_x0, src_dx, src_nx,
    src_y0, src_dy, src_ny,
    src_z,
    rec_x0, rec_dx, rec_nx,
    rec_y0, rec_dy, rec_ny,
    rec_z,
    ORIENTATION,
    moveRec)

if not status:
  raise RuntimeError(f"Unable to generate geometry: {out_geom_name}")

# add geometry container to treeview
slicer.util.mainWindow().emitH5FileToBeAdded(geomcnt.getH5File().getFileName())

# don't forget to close h5geo (hdf5) objects
del geom
del geomcnt
```

Source geometry along x-axis contains 8 sources at positions from 1500 to 2200m with 100m step. Along y-axis: from 1000 to 1500m wtih 100m step. Receivers are spread in in 1000m*1000m (i.e. sp+-500m) with 50m step and they are moved along with sources (`moveRec = True`).

Geometry check

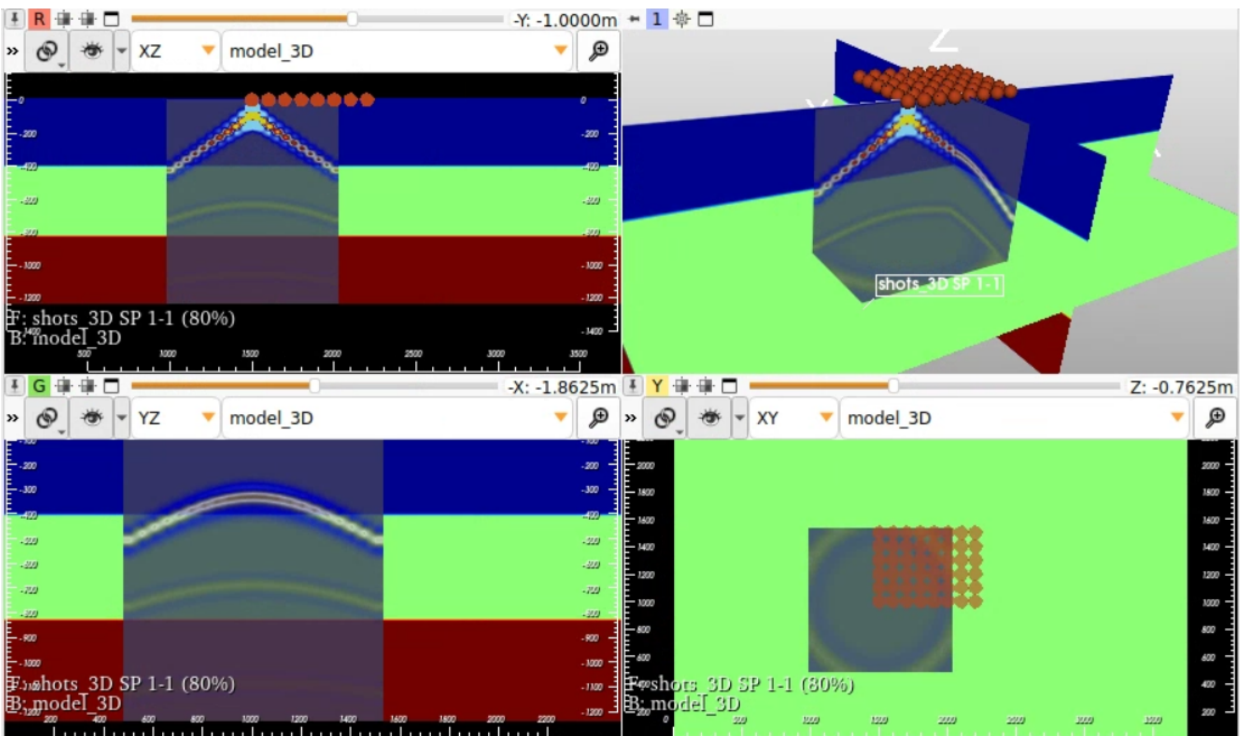
We can load source and receiver positions for each shot. For example for the first shot we can see this:



Red marker shows source position for all these receivers.

5.3.3 Forward Modeling

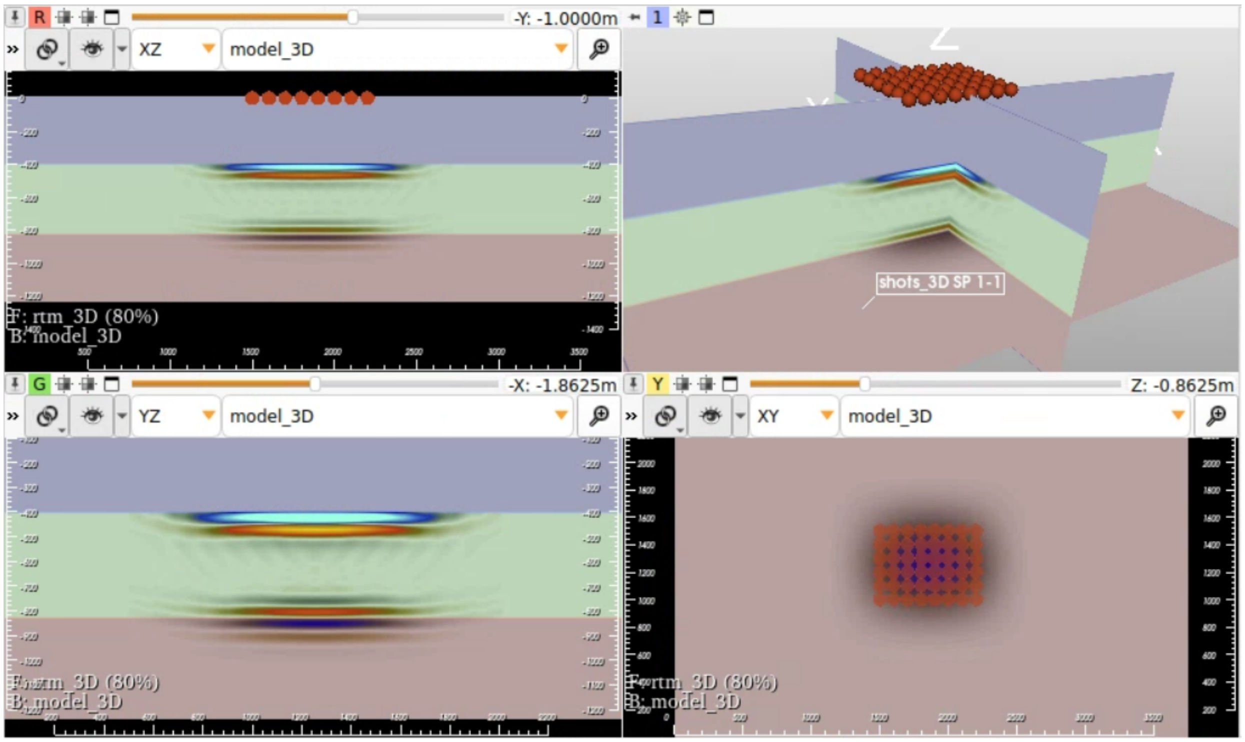
In `Wave Modeling` module set all the parameters that were used in *Simple Wave Modeling 2D tutorial*. It is also necessary to set *Limit modeling domain* to at least 1000m. This setting defines the region of the model that is used for every shot computation.



5.3.4 Reverse Time Migration (RTM)

Even if we set *Limit modeling domain* to 0m then **30Gb of RAM** is required for RTM. Actually expanding this area doesn't greately improves the RTM quality. As in *2D wave modeling tutorial* we have to set model muting to 412.5m and data muting to mute turning waves.

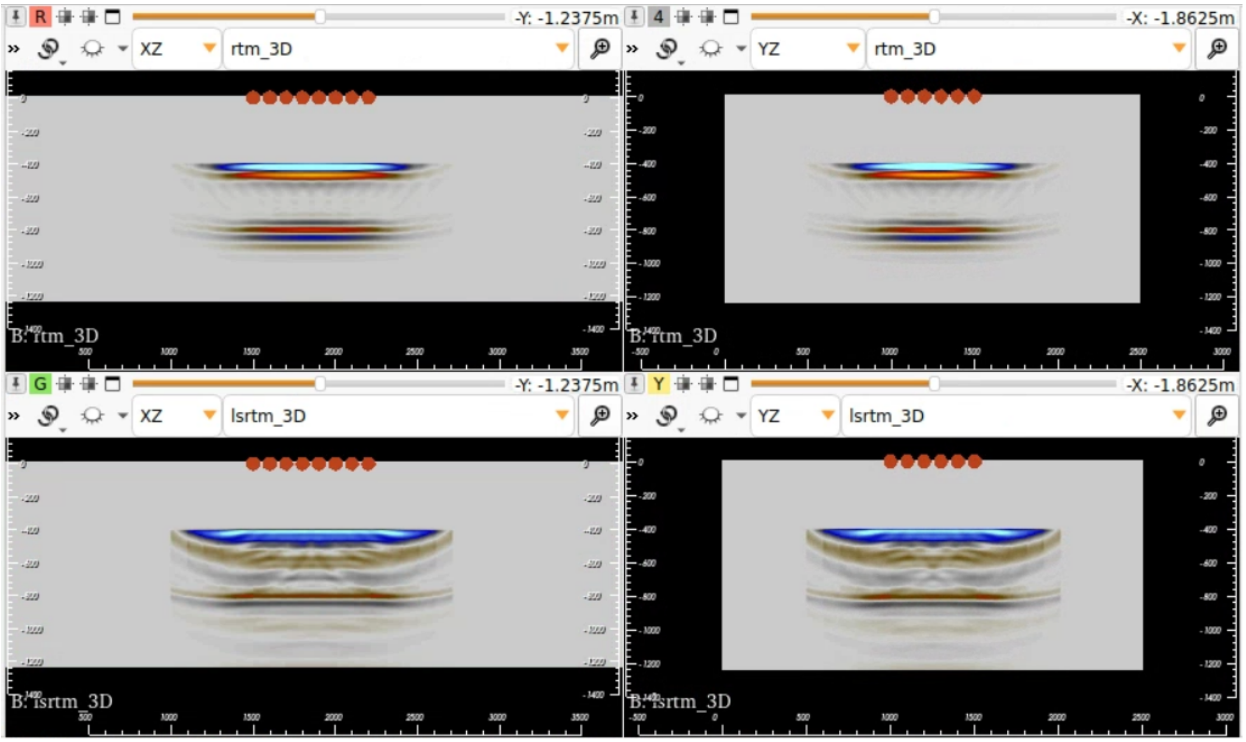
Velocity model overlayed with RTM:



5.3.5 Least-Squares Reverse Time Migration (LSRTM)

To run LSRTM about **30Gb of RAM** is required.

Select true velocity model as *Input*. *Geometry* same generated shots. Use same mutings that were used in conventional RTM.

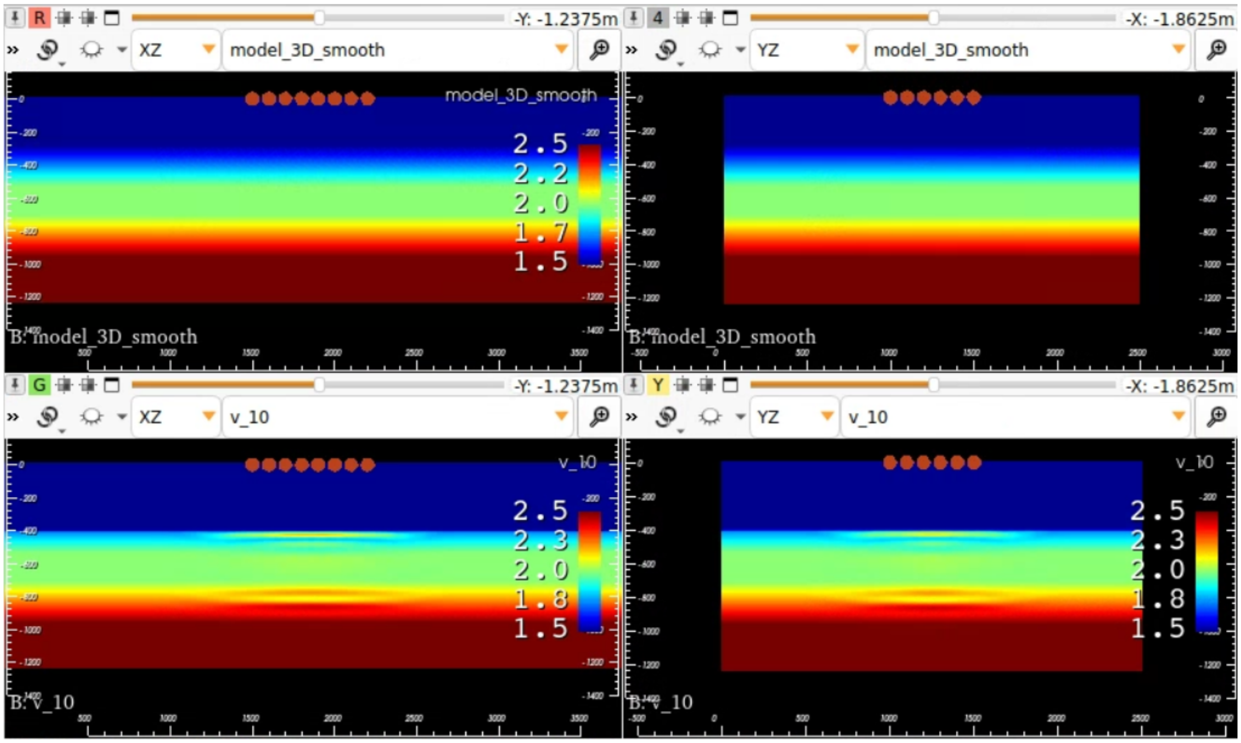


Upper pictures show conventional RTM. Lower - result LSRTM.

5.3.6 Full Waveform Inversion (FWI)

FWI with *Limit modeling domain* to 0m also requires **30Gb of RAM**. Don't forget set starting model *model_3D_smooth* as input. **Important thing** is to turn on *Replace water velocity/density*. This will replace velocity (and density if present) at each iteration.

In *FWI Settings* set min/max velocities to 1000 and 3500 respectively.



Upper pictures show starting smoothed model at XZ and YZ orientations. Lower - result of 10th FWI iteration.

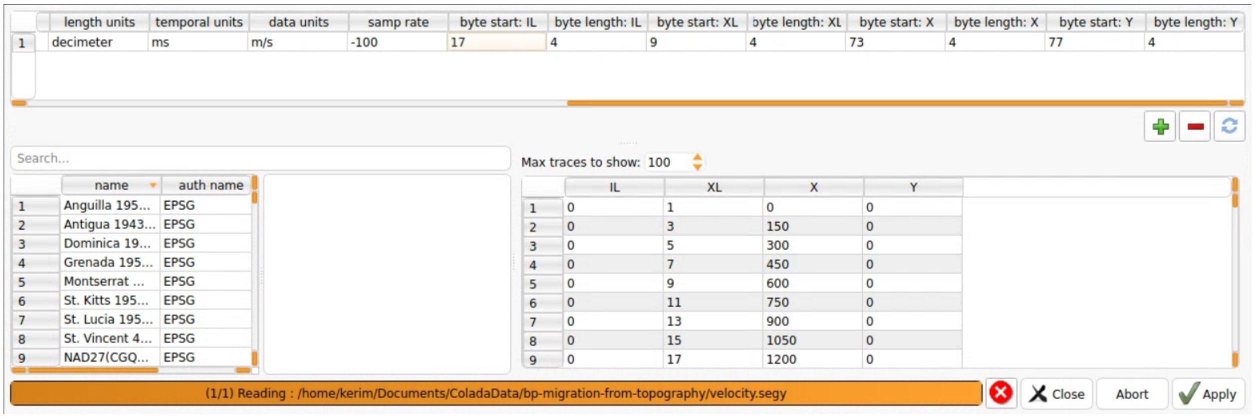
5.4 Wave Modeling: 1994 BP migration from topography dataset

Before getting started one should download velocity model from SEG page.

5.4.1 Reading data

Velocity model is represented by SEGY file. One should read it as Geo Volume: right click on *Geo Volume treeview* -> *Import* -> *SEGY STACK*. Read parameters:

1) Domain: *TVD*
2) Length units: *decimeter* (use this service to check whether units can be converted to any other)
3) Data units: *m/s*
4) Samp rate: *-100*
5) Byte start: IL: *17*
6) Byte start: XL: *9*
7) Byte start: X: *73*
8) Byte start: Y: *77*



Warning: As some trace headers are incorrect in this SEGY (see few last traces) most likely volume parameters (origin, spacings) should be manually fixed then. For that after SEGY is read go to the *Geo Volumes* module and select volume read. Make sure that origin is [0,0,-99900] and spacings are [150, 1, 100] decimeters. After setting this and clicking *Update* button the velocity model is ready.

5.4.2 Generating geometry

From data description geometry should contain 277 shots with 90m interval. 480 traces per shot spread from -3600m to 3600m with 15m interval. Receivers are moved along with source. Sampling rate is 4 ms, number of samples 1000.

In Colada geometry is Seis object. Thus go to the *Seismic* module and create new seismic with parameters:

1) Data type: *PRESTACK*
2) Survey type: *TWO_D*
3) Domain: *TWT*
4) Number of traces: let it be *480* (will be changed anyway)
5) Number of samples: *1000*
6) Sampling rate: *4*
7) Trace chunking: *480* (see HDF5 chunking)
8) length units: *m*
9) Temporal units: *ms*

Click *Create* button.

Then in the *Geometry* section set:

1) Source geometry x0,dx,nx: *0,90,277*
2) Receiver geometry x0,dx,nx: *-3600,15,480*
3) Check on *move receivers with sources*

Click *Generate Geometry* button.

5.4.3 Forward modeling

Go to the *Wave Modeling* module. Set velocity model as input. Generated geometry as *Field Records (Geometry)*

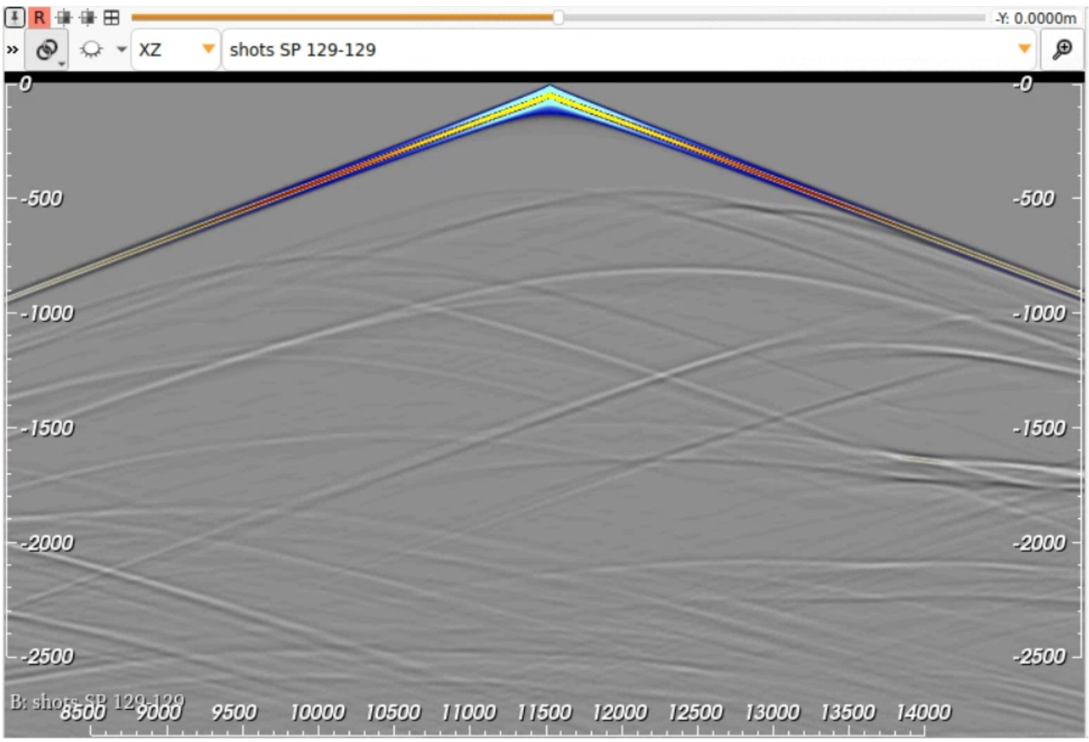
Modeling settings:

1) Computing type: *Forward modeling*
2) Ricker frequency: *0.02* kHz
3) Select *Save file prefix* and *Output dir*

Click *Start Computing*.

Note: It is recommended to always compute one or small number of shots before starting computation on all shots. This allows to check if the solution is accurate enough or not and make changes to settings. For that in *Additional settings* set *Compute each Nth shot* to some big number.

After computations are done one can see the result, for example this is 129 shot:



5.4.4 Reverse Time Migration (RTM)

RTM works with field data thus geometry input file must be replaced by the computed synthetic records.

Then in *Common computing settings*:

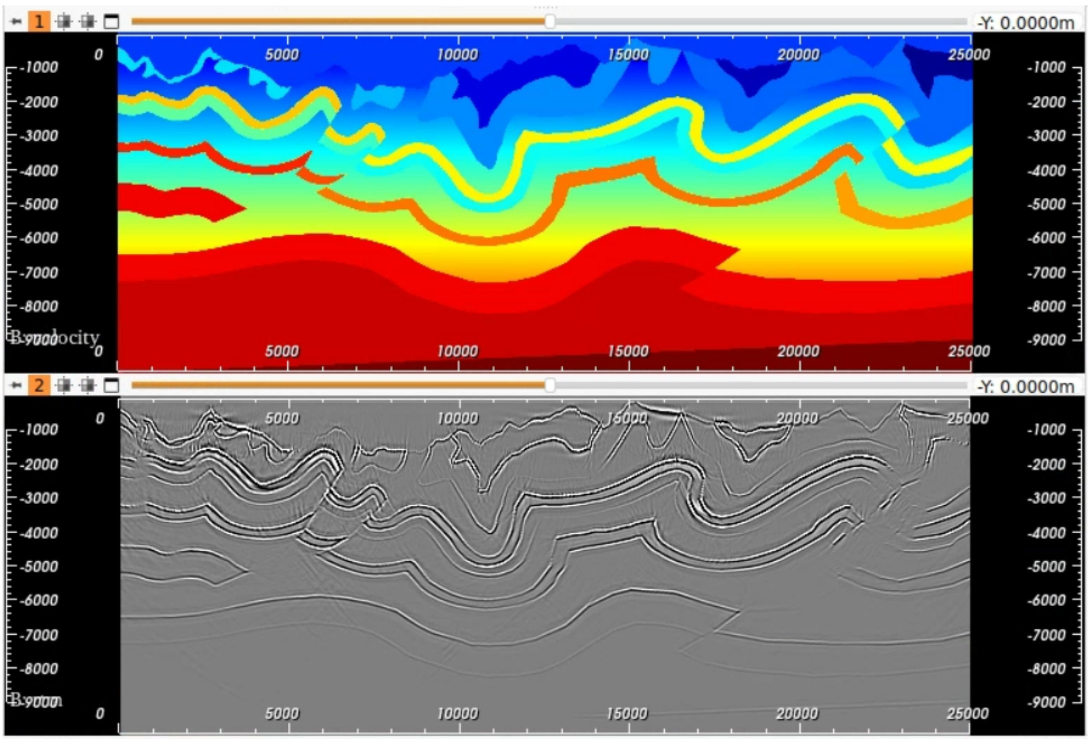
1) Computing type: *RTM*
2) Limit modeling domain: *0* (if set to 3000 then about 35 Gb RAM is required)
3) Change *Save file prefix* and *Output dir* (the result will be Geo Volume)

and in *A*dditional settings*:

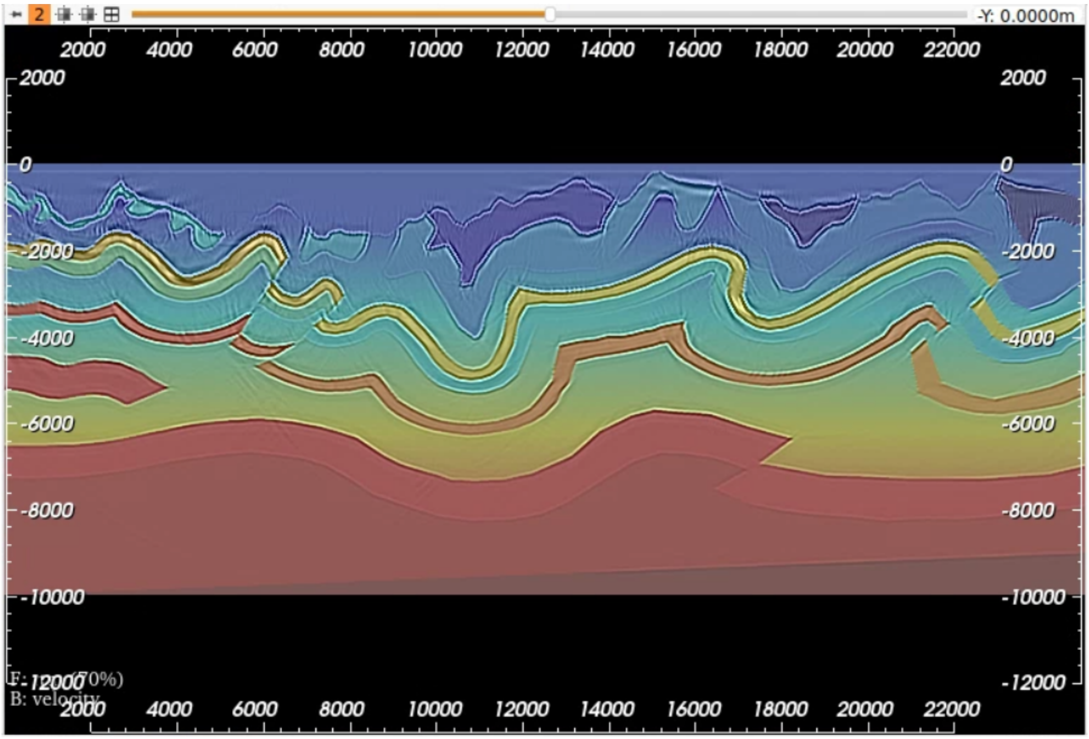
1) Mute model: *On*
2) Mute model (m): *170*
3) Mute model taper (m): *10*
4) Mute data: *mute turning* (upper triangle mute)
5) Mute data start time (t0,ms): *140* (approximate length of a 20 Hz Ricker signal)
6) Mute data velocity (km/s): *4*

Click *Start Computing*

The result may be similar to this:



and this is velocity model overlayed by RTM:



Hope you liked it.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search