Chapter 4

MODEL SETUP AND VALIDATION

4.1 Introduction

Hetland et al. [2010] (referred to as HSD10, henceforth), and Hetland and Simons [2010] (referred to as HS10, henceforth) developed a procedure for simulating slip evolution on a planar fault surface resulting from kinematically imposed ruptures on pre-defined portions of the fault ("asperities") that are otherwise locked during a seismic cycle. Slip evolves on the fault surface because as the stress perturbation around an asperity resulting from the imposed seismic rupture decays. The rate of this decay is determined by the induced slip-rates, which in turn, are determined by the fault rheology. A fundamental feature of this model is that the mean stress on the fault surface evolves to a steady state because the imposed ruptures and far-field loading are identical over multiple cycles. This "spin-up" results in self-consistent background stresses that depend only on the history of past ruptures. The above formulation supports a variety of rheological relationships (linear/non-linear viscous, purely rate dependent, or Dietrich-Ruina rate and state friction).

The work presented here extends the procedure developed above to not only handle 3D fault geometry and actual rupture history for a given megathrust interface, but also the complex visualization required for the analysis of fault surface parameters, surface velocity field, as well as routine quality control checks. Several major developments were required in order to achieve the above goals:

- HSD10 and HS10 use a planar fault discretized into rectangular patches. However, it is nearly impossible to discretize an arbitrary 3D surface using rectangles without developing surface discontinuities ("kinks") in regions having even moderate curvature, unless extremely high resolution is used over the entire fault surface. For the problem to be tractable, we require a mesh that is fine only near the asperities,

where the gradients in the stress and strain fields are extreme. For such an "adaptive" mesh, kinks are unavoidable, unless extra elements are added wherever there is non-negligible curvature along the fault surface. Since we are dealing with the decay of coseismic stress "pulses" along the fault surface, the stress-singularities resulting from such kinks would not only dominate the simulation, but also make it very unstable. Here, we choose to avoid such kinks by discretizing with triangular patches. We use a comprehensive geometry and meshing package developed by Sandia National Laboratory, Cubit [Owen, 2006], for this purpose.

- The discretization above is a four-stage process: First, the geometry needs to be generated in a CAD package such as the widely used commercial package for geologic applications, Gocad [2010]. Such a base model was put together by Eric Hetland. Next, this geometry needs to be imported into a meshing package like Cubit, cut and smoothed into an orthogonal edged curved surface. Then, the smoothed fault surface is discretized using an optimal adaptive mesh (different for every configuration of asperities) designed to focus most elements ("patches") around the asperities. Finally, the numerical quality of the mesh and its smoothness are improved through an iterative process involving the computation of fault traction kernels at the centroids of all patches (see below).

- Before this project, the only openly available triangular dislocation solutions [Meade, 2007] were in Matlab and much slower than the Okada [1992] solutions available for rectangular patches. So, effort was put into improving the efficiency of the triangular solutions (by 250X), building a common interface for both types of solutions for benchmarking purposes, so they can directly process the discrete patch information from unstructured surface meshes output from Cubit. Fault traction kernels at the centroids of all patches are used to determine the local mesh resolution, and hence, the quality of the mesh (as described in the previous bullet-point).

- HSD10 and HS10 use Matlab for visualizing the simple planar faults tested in those papers. Since here, we are dealing with unstructured meshes for 3D fault surfaces,

arbitrarily oriented surface grid for computing synthetic observations, and a "point-cloud" of stations at the surface where actual data is available, a visualization system is needed that can handle such complex tasks on-the-fly. For that purpose, we use a comprehensive, open-source, 3D visualization package, the Visualization ToolKit [Schroeder et al., 2006], VTK, that allows easy visualization of multiple types of 3D spatial datasets such as those mentioned above, as well as their temporal evolution. Such visualization is an essential part of both the analysis of simulation output as well as routine quality control tasks such as checking and validating model input parameters.

- HSD10 and HS10 do not consider surface displacement/velocity fields, but instead focus on the stress shadow effect on the fault surface due to different configurations and shapes of asperities. Here, we want to compare the surface velocity predictions from the model to actual GPS data. In order to facilitate this, we needed to not only make sure the model is properly geo-referenced, so correct station locations can be used, but also develop a way to view maps and geographical locations (e.g., epicenters) for visualization in both a meshing package like CUBIT as well as VTK-based visualization packages (e.g., Paraview, and Mayavi). Without such geo-referencing, it is hard to make sure the locations of asperities are consistent with the epicentral coordinates, and their extents are consistent with say, topographic features on the overriding plate (e.g., locations of long-term geologic uplift, locations of coastlines, undersea canyons, etc.).

- HSD10 and HS10 use one or two asperities with relatively simple rupture sequences with specified rupture intervals on each asperity. Model spin-up was a straightforward point measure — namely, the mean value of tractions along the fault surface at the last time-step of every cycle. Here, we are dealing with a real subduction zone, where the rupture sequence for a given set of asperities is quite arbitrary, and the rupture interval has to be estimated from the best known near-field seismic and geodetic data. Also, in such cases, the definition of a "cycle" for a single characteristic earthquake has to be changed to that of a "cycle" for a characteristic

sequence of earthquakes — a characteristic rupture sequence time, or CRS-time. Instead of a single point in time, spin-up has to be tested over a span of at least a single CRS duration. In order to check for spin-up of solutions, the simulated tractions are averaged over several moving average "windows" and the smallest window (typically, the CRS-time) that shows stable spin-up is chosen to determine which cycle to use for finally computing synthetic GPS velocities to compare to data.

- HSD10 and HS10 use all asperity patches in their forward calculations. Here, by predefining the asperities during meshing, we compute fault parameters only on fault patches during forward modeling. Given that a large fraction of the total number of patches lie within asperities, we end up with up to 50% smaller kernel matrix sizes for a given mesh resolution (see below), or use 50% more elements for modeling stress/slip evolution on a given fault surface.

This chapter details the framework of pre-processors, elastic-dislocation codes, and post-processors that were developed to handle simulations with realistic 3D fault surfaces. In what follows, the general codes we developed for generating a smooth megathrust interface, its discretization into patches for kernel and model parameter generation, and major developments to the existing forward model solver to handle the complexities of 3D faults as well as realistic rupture patterns, as well as some key post-processing steps are discussed. While each of the above steps is "automated", the underlying code was built so that the results from each stage can be extracted, visualized, and validated independently. All newly developed codes use open-source and/or free programming languages/packages. All lower-level (and stable) code is built as a python package ("Fslip") which is installable, so it can be invoked from any python script or dynamically, in the python interpreter.

Figure 4-1 illustrates the workflow involving the *Fslip* package and external packages used for the analysis here. Currently, there are two components to running a model using *Fslip*: The python pre- and post-processors developed here, and the forward model solver

in Matlab (*EvolveSlip*, written by Eric Hetland, and modified for more general application to real faults here).

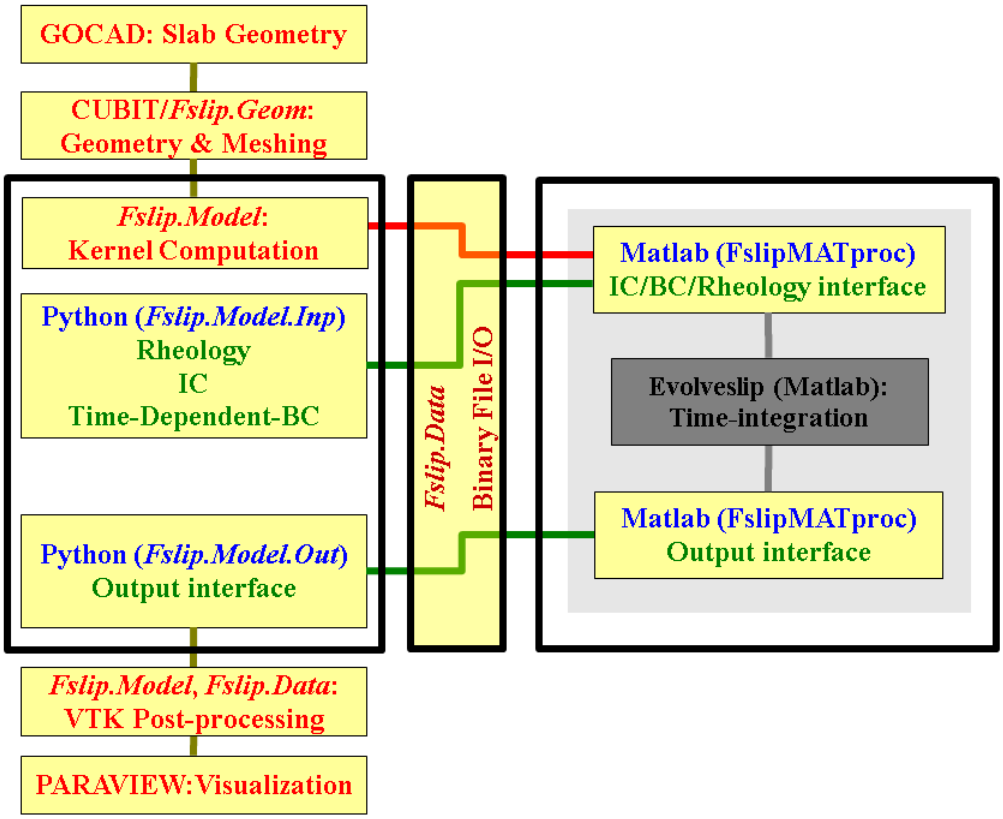


Figure 4-1. Workflow for the *Fslip* software developed to generate, simulate and visualize models of slip evolution on realistic fault surfaces. For simplicity, the exact long-from module names are not used for the lower level modules or Matlab pre-/post-processor scripts (e.g., *Fslip.Model.Out, FslipMATproc, etc.*). The matlab component, *EvolveSlip*, solves the forward problem, given the fault traction and surface displacement kernels, rheology, initial and boundary conditions for the 3D megathrust surface. The Matlab output is post-processed in two stages — in a Matlab and then in *Fslip* to generate VTK files and "dashboards" for visualization. See text for details.

4.2 Model visualization

Given the complex nature of real fault/station geometries, it is important to have the ability to probe both input and output data "on-the-fly". As mentioned previously, we use VTK, as it is an extensive 3D data visualization package having well-defined native input file formats for different data types (e.g., structured/unstructured grids, and point

clouds). For example, while the fault is discretized as an unstructured mesh, the structured mesh is used to define a surface grid of observation points to sample surface deformation more densely than station locations allow. Stations are treated as surface point clouds. The XYZ-Cartesian system in which the 3D fault interface is defined in a CAD package such as Gocad [2010], becomes the "geographic" reference frame for all the forward models: i.e., all hypocenters, stations, and seismic asperities on the megathrust interface are now defined relative to. this reference frame in the VTK format. Coastal boundaries and trench location are also defined in this frame for visually locating the extents of seismic asperities on the megathrust interface. These legacy or XML-based VTK files can then be visualized using several open-source or free software packages (Mayavi, Paraview, etc). Such "on-the-fly" visualization is extremely helpful in debugging model inputs as well as outputs by making it possible to quickly probe the quality of a 3D model or simulation statistically, point-wise, along line-profiles, cross-sections, as well as from different angles. The tremendous flexibility in visualization, however, comes at the cost of extensive bookkeeping, and file I/O required for keeping track of the multiple types of data-sets being handled here. So, both pre- and post-processing steps are more involved and much slower than the Matlab based plotting used in HSD10 or HS10. While VTK follows a relatively simple protocol for mesh numbering (e.g., nodes are numbered in the order they appear in element connectivity), translators are required to convert unstructured mesh nodal IDs (which are typically output in arbitrary order by Cubit) into the ordering expected by VTK. All this functionality is built into the input and output modules under *Fslip.Model*. The actual file I/O is performed under the *Fslip.Data* module.

4.3 Megathrust interface geometry & discretization

Since triangular patches can represent a 3D surface with higher accuracy than quadrilateral patches, we choose unstructured triangular meshing to divide the megathrust interface into discrete patches. All model parameters (i.e., rheological parameters, stresses, slip, slip-rate, any state-variables, etc.) are defined or computed at the centroids

of these triangular patches. The procedures adopted for creating and analyzing the megathrust geometry and subsequent triangulation into patches are discussed here.

Table 4-1 illustrates our workflow for generating an optimal discretization of the fault. The geometry for the upper surface of the subducting slab for the whole of Japan was generated by Eric Hetland, by fitting hypocentral locations, and published seismic reflection surveys in the commercial geological CAD package, Gocad (Table 4-1.1). These discrete patches ("facets") are then exported to the geometry and meshing package, Cubit [Owen, 2006] (Table 4-1.2). All stages in the geometry/mesh generation process are fully automated using python scripts within CUBIT, using separate input files for model geometry and meshing. These functions are available under the module *Fslip.Geom*. However, at the present time, access to Cubit without the GUI interface ("Claro") is not straightforward (since CUBIT is not open-source, the version of python packaged in the binary release might conflict with the local system python), so the default assumption behind these geometry/mesh scripts is that they will be run from inside Cubit's python command line interpreter.

In CUBIT, the geometry is first cut along a vertical surface defined by the trench profile. The same profile is shifted down-dip (by ~ 400 km), to make a parallel cut defining the bottom of the "active" megathrust boundary, or "fault". Two vertical surfaces, locally orthogonal to the trench-profile then cut the interface, completing the basic fault geometry generation. The resulting orthogonal-edged fault surface is extended to the surface (z=0), then smoothed to remove any along-dip kinks in the fault surface, owing to the fact that the amount of updip extension required to get to the free-surface varies along strike. Next, the fault surface is extended along the local trench strike at both ends, so its boundaries are far-enough from the outermost asperities (~ 4–5 times the characteristic asperity size, D_{asp}). The surface is smoothed once more after this extension to remove any kinks along strike (Table 4-1.3). This two-stage smoothing of the fault surface was found to be necessary in order for the tractions computed from any triangulation (or mesh, see below) to be relatively smooth over the curved fault surface. Next, the fault surface is subdivided into six orthogonal edged surfaces (three along strike, times two

down-dip), such that the upper middle surface ["Fsurf", the yellow patch in (Table 4-1.4a)] contains all asperities. "Fsurf" enables tighter control over transitioning of the mesh from coarser resolution near the fault boundaries to very fine resolution around the asperities (patch sizes vary over two orders of magnitude). The asperities are then cut out of "Fsurf" and locally smoothed around each "cut", to remove any discontinuities. The megathrust interface is now ready to be meshed (Table 4-1.4b).

The first step in the meshing process is setting interval sizes for its bounding curves and/or surfaces. The interval size used directly affects not only the quantity, but also the quality of elements in the mesh. We set the mesh interval size for the asperity-region bounding curves, as well as all asperity boundaries, to ~ 0.3-0.5 of the interval size for rest of the fault. This ratio of interval sizes depends on the aspect ratio of "Fsurf" relative to that of the fault surface, as well as the circumference of the smallest asperity relative to the edge-lengths of "Fsurf". We use "QTri" patches (quads divided into constituent triangles) outside the asperity region, as these were found to result in much faster convergence of the forward model (as opposed to fully unstructured triangular patches). Meshing within "Fsurf" as well as inside the asperities is done by an automatic paving algorithm built into Cubit.

The mesh is first refined (each triangular patch is split into three triangles formed by its edge-bisectors) around to asperity, to improve its resolution at a coarse level. So, it is important to obtain good quality elements around the asperities in the lowest resolution mesh because any poor quality element yields poorer quality "child" elements during such refinement. The whole of "Fsurf" is now refined once, transitioning into the coarse "QTri" elements outside this asperity region. This forms the base resolution mesh (~3500 fault elements, and ~ 500 asperity elements, for the mesh presented here), "RES0" (Table 4-1.5(a)). The mesh itself is smoothed after every refinement step, to eliminate any abrupt element-size changes as well as better represent the original geometry. Since our model is driven by near singular tractions due to ruptures on asperities, most of the driving stress is concentrated right around the asperity. Therefore, further refinement is intended to better resolve the stresses around the asperity, and

results in a transition zone of high mesh density between the asperity and the surrounding fault surface. The upper limit on the number of elements is placed by 64-bit Matlab's restriction of 10^4x10^4 elements. This limitation can be overcome when the forward model code, *EvolveSlip*, is eventually ported out of Matlab into Python. Given this restriction on the maximum number of fault patches, there are tradeoffs between the dimension of the smallest asperity, the fractional area of "Fsurf" that all asperities constitute, the size of "Fsurf", and the number of elements in the transition zone between the asperity and "Fsurf" – "Ftrans". For the coarse "RES0" mesh, roughly one-fourth of the fault patches (that is, excluding asperity patches) lie within a distance from asperity centers equal to the respective asperity dimensions ("Ftrans"). For the highest resolution mesh tested here ("RES2", with ~9,500 fault elements, and 4000 asperity elements), nearly 70% of the fault elements lie within this same distance (Table 4-1.5(b)). Thus, virtually all of the increase in resolution (~ 6000 elements) occurs right around the asperities, leaving the rest of the mesh relatively untouched. One could generate meshes that conform to the fault surface outside of "Fsurf" even better, by adaptively focusing on regions of higher curvature. In our case, the added expense due to refined elements far-from the asperities did not allow for the use of such meshes. However, this form of mesh refinement can be explored in the future.

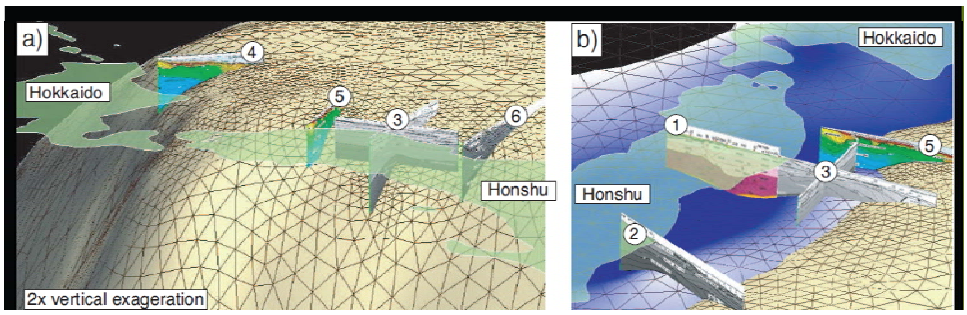
The ultimate check for a mesh is the smoothness of the traction kernels computed from that mesh. For every mesh used in benchmarking and simulations, several iterations were required to remove any subtle kinks from the fault surface. The goal was to preserve the large scale 3D structure of the fault, without introducing numerical perturbations of the traction field (at least in the predominant slip direction). Knowledge gained from these iterations were directly incorporated into making the geometry and meshing modules, *Fslip.Geom*, more effective. Benchmarking with kernels is discussed in more detail below.

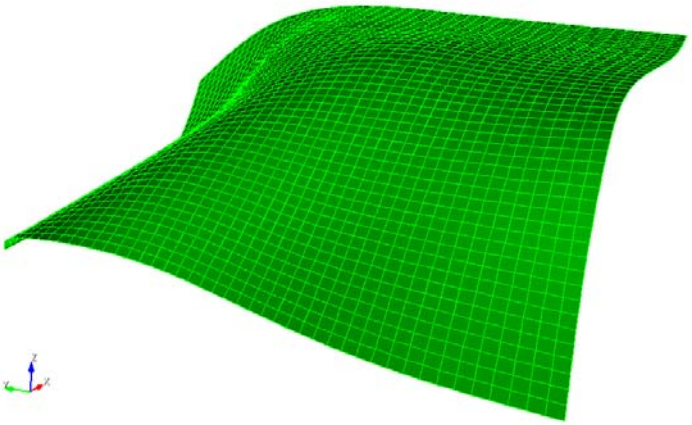
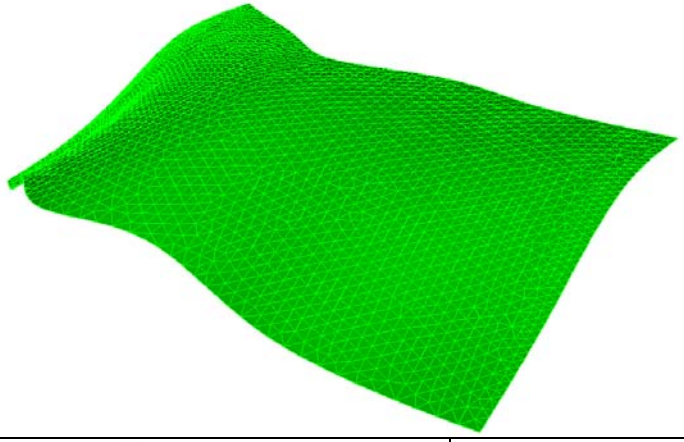
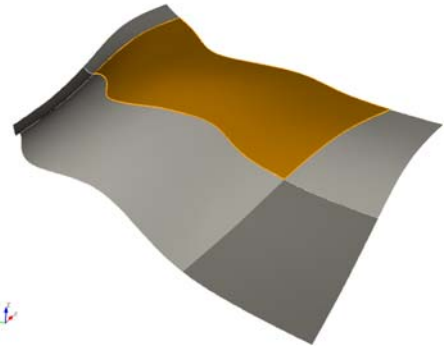
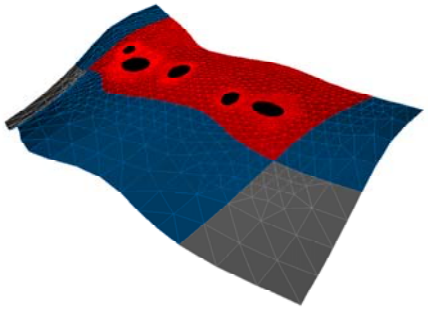
In order to improve the accuracy of locating asperities relative to the geographical region, a set of scripts were written to import a coastline map into the meshing package, Cubit. This not only aids in visualizing where the asperities are relative to surface stations, but

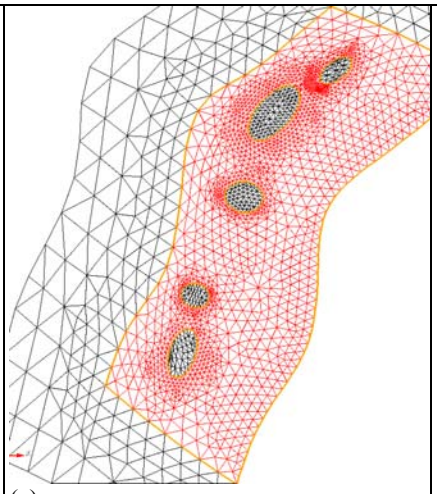
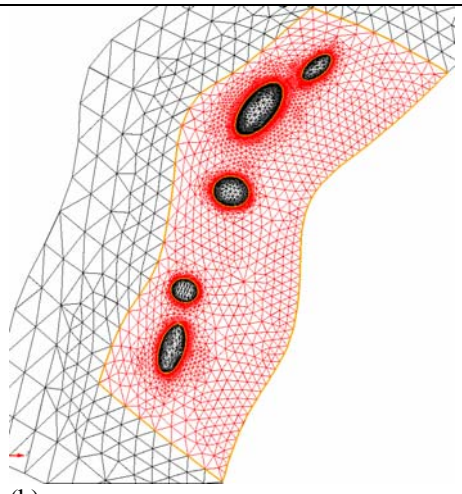
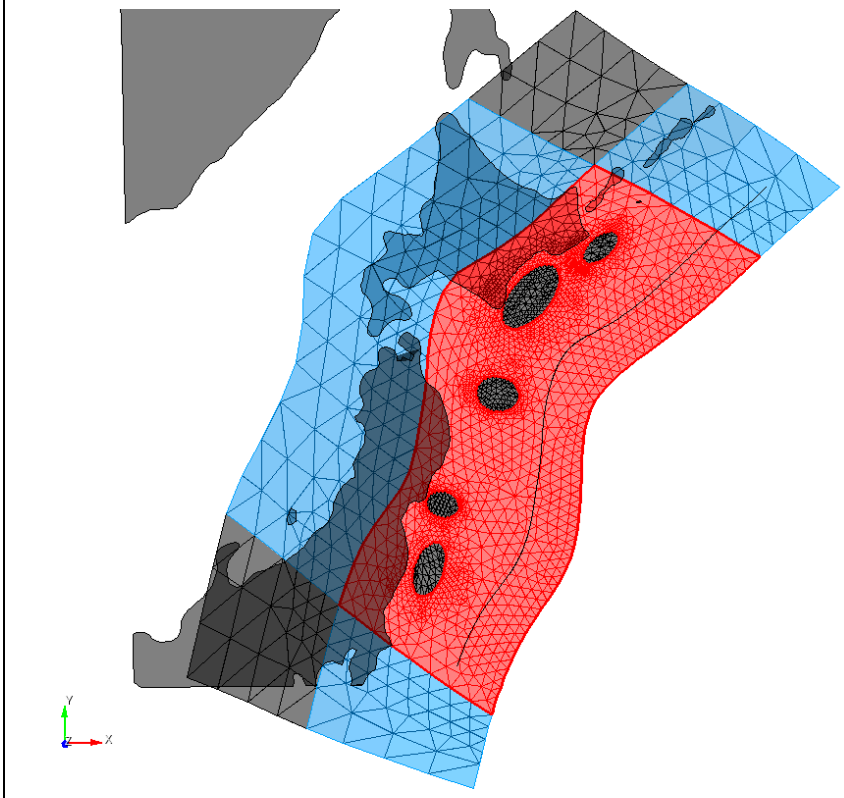
also provides a scale for quickly checking their extent, as well as their location relative to seismic epicenters (Table 4-1.6).

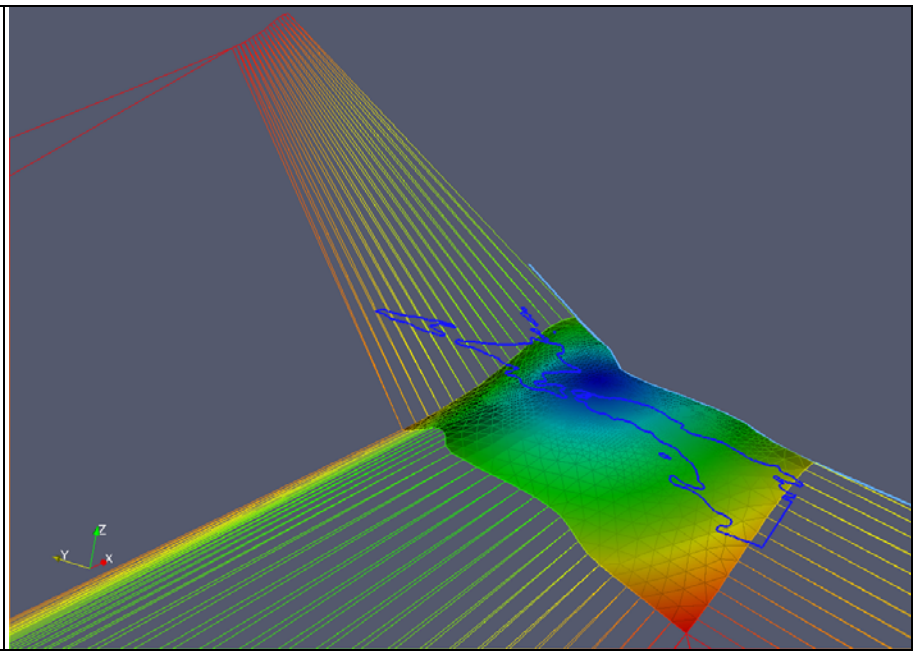
Finally, the mesh is output in Abaqus as well as ExodusII (for the sake of generality) formats. An Abaqus parser module (within *Fslip.Data*) and a comprehensive mesh post-processor module (within *Fslip.Geom*) were written to then extract patch data for Kernel computation. Such patch data include the nodal coordinates, area, aspect-ratio, strike, and dip of each element. Given we are dealing with unstructured meshes here, the mesh post-processor also includes several advanced utility functions to generate semi-infinite extensions of the curved fault surface (for imposing far-field boundary conditions), finding neighboring elements based on several criteria, e.g., shared edges, or element IDs, are included in this module. In addition, the mesh post-processor is general enough (and tested) to handle both triangular and quadrilateral meshes. The output from the Mesh post-processor is directly used by *Fslip.Kernel* module, to compute any type or combination of Kernels, as discussed in the next section.

Table 4-1. Work-flow for generating fault patches from a geo-referenced subducting slab surface geometry.

GEOMETRY	
1. GOCAD Facets	 Models of the slab surface (a) and the Japan Mohorovicic discontinuity (b); for reference, a few of the seismic lines used as constraints are shown: 1, Iwasaki et al [2001]; 2, Miura et al.[2003]; 3, Ito et al. [2004]; 4, Nakanishi et al. [2004]; 5, Takahashi et al.[2004]; 6, Miura et al. [2005]. (Above figure, courtesy Eric Hetland)

2. Cubit ACIS net-surface approximation to Gocad facets	
3. Smoothed, orthogonally cut fault surface generated from the above mesh	
4. Subdivided fault surface (a) creating an upper-central area for locating asperities, "Fsurf" [yellow region in (a), red in (b)]	 (a)  (b)

MESH		
5. Coarsest ($4x10^3$ patches) and finest (10^4 patches) mesh resolutions used for testing convergence of solution	 (a)	 (b)
6. Adaptively meshed fault surface with asperities, along with a coastal map (only coarse resolution shown here) of Japan, and epicentral locations. Such meshes with map overlays can be routinely generated in "auto" mode.		



7. Semi-infinite extensions of the fault surface, that can potentially be used to test the validity of backslip on the active fault. (Colors indicate centroidal distance from the center of the projected Cartesian reference frame)

4.4 Kernels

The term "Kernel" is used here to denote the matrix of impulse-response functions that define the tractions or displacements at each observation point due to a unit slip along each of the constituent source patches of the fault. Considerable effort was expended in generating and testing the source code for the kernels. The existing half-space solutions for rectangles [summarized in Okada, 1992] and triangles [summarized in Meade, 2007] have different sign conventions. The Okada [1992] solutions were originally written in F77, but their most common form is a Matlab-MEX file, which needs to be recompiled and/or modified with each Matlab upgrade. The Meade [2007] solutions were entirely cast in Matlab, but very inefficient because the bulk of the calculations were "cut-and-pasted" from Matlab's symbolic language toolbox. The Meade [2007] approach resulted in typical arithmetic expressions that were thousands of elements long, with nearly as many repetitions of function evaluations. Significant improvements in efficiency could be achieved by simplifying these symbolic language expressions. Here, triangular patches are used to accurately represent a real (curved) megathrust interface, and the much widely used rectangular patch solutions were used to benchmark these triangular solutions.

In order to overcome (a) the lack of a common interface for defining real-world faults for either the Okada [1992] or Meade [2007] solutions (the existence of which would be useful for benchmarking), (b) improve the computational efficiency of the triangular-dislocation solutions, and (c) make these solutions available in open-source format, two distinct but architecturally similar python-wrapped packages (*OkadaWrap*, and *TriWrap*) were developed, having identical functional interfaces that could be called from the same driver script using identical input data (a set of fault patches and observation points). These packages were extensively benchmarked using a suite of tens of 2D and 3D test problems — not only against each other ("peer-to-peer"), but also against the original Matlab versions ("parent-to-peer"). For *OkadaWrap*, the original F77 code was converted into F90 syntax, while for *TriWrap*, the original code was entirely re-written as optimized F90 modules. For *TriWrap*, special python scripts were written to automatically parse and translate the hundreds of Matlab expressions (output from the symbolic toolbox) into F90 syntax, as well as subsequently simplifying the resulting expressions to eliminate any repeating expressions or intrinsic function calls. As a result, *TriWrap* was found to be roughly 250 times faster than the original Matlab code [Meade, 2007] during benchmarking. The core dislocation codes in both packages are supported by several common F90 modules. The respective F90 driver routines for each package take care of the different input parameter conventions for these two solutions, and assemble kernels for a specified combination of patch and observation stations. These F90 driver routines are finally wrapped into python functions using the *F2py* package within *Numpy*. Installing and using either of these packages require both python/numpy as well as one of the free F90/95 compilers (Intel Fortran, or GFortran), but scripts for compiling with *F2py* are included with the packages. Beyond this basic requirement, the end-user can simply use these solutions as a "black-box" with a common interface. The packages also provide sample "well-commented" python driver scripts that can be a good starting point.

While space does not permit presenting the tests done in ensuring the accuracy of the kernels, it is worth mentioning the kinds of tests that were carried out in comparing the two sets of kernels. In all tests, fault orientation (strike and dip) were varied to get four

different combinations (positive or negative values for each angle). 2D tests included comparison of both free-surface and along fault displacement and traction vectors, as well as the full stresses tensor, for different fault and observation profile resolutions. The profile along the free-surface or fault was varied to compare both near- and far-field predictions. 3D tests involved comparing surface and fault deformation fields due to slip on a planar fault, discretized using rectangular patches, with a planar fault discretized using different triangular subdivisions of these same rectangles (from two to seven triangles per rectangle).

For the Japan megathrust interface used in the simulations here, the computed traction kernels were checked for both internal consistency and smoothness. In the context of half-space models, if a fault surface were to rupture the entire half-space, the relative motion between the hanging-wall and foot-wall can be described by rigid blocks sliding past each other. There is no net strain accumulation anywhere in the half-space owing to such rigid-body sliding. Say, we divide this fault surface into three regions: (a) small patches that slip only episodically ("asperities") during seismic ruptures, (b) a region surrounding these asperities that slips in response to these ruptures ("fault"), and (c) the semi-infinite extension of the fault both along strike and dip ("loading"). The basic idea here is that over the course of an appropriately defined seismic "cycle", all three regions of the fault would experience the same slip (albeit at different times), so that the net motion is equivalent to rigid-block motion. Let us now define at every point on the fault surface (in practice, at the centroid of every triangle on the discretized fault), (a) tractions resulting from slip on all fault patches (excluding asperities), τ_{FF}, (b) tractions resulting from slip on all asperity patches, τ_{FA}, and (c) those resulting from slip of the semi-infinite loading patches that continuously slip at the plate convergence rate, τ_{FI}. If all three regions slip the same amount, then the resulting rigid-body motion implies that at every point on the semi-infinite fault surface within the half-space,

$$\tau_{FF} + \tau_{FA} + \tau_{FL} = 0 \, . \qquad (1)$$

For realistic 3D meshes, however, discretization of the fault surface results in not all components of Equation 1 exactly cancelling each other. This consistency check is carried out every time kernels are computed for a given asperity configuration. "Dashboards" containing 3D views of the above fault tractions are automatically generated for viewing in the open-source parallel VTK visualization package, *Paraview* [e.g., Henderson, 2004]. This consistency check (Equation 1) is presented for each traction component (strike-slip, dip-slip, and normal), for dip-slip over the fault, asperity, and loading patches in Figure 4-2, and for strike-slip along these patches in Figure 4-3. In these figures, each row of panels represents the contributions for one traction component, while each column represents τ_{FF}, τ_{FA}, τ_{FL}, and the residual $\Delta\tau$ (the right-hand side of Equation 1), respectively.

Since coseismic tractions are the principal driver of the models presented here, both the 3D fault geometry and the mesh were iteratively smoothed until the tractions along the principal slip-component directions (e.g., τ_{dip} due to dip-slip, and τ_{strike} due to strike-slip) exhibited local fluctuations of $< 0.1\%$ of the main coseismic signal around the asperities. However, for the "cross"-component directions (e.g., τ_{strike} due to dip-slip, and τ_{dip} due to strike-slip), these fluctuations cannot be entirely eliminated owing to the fact that these tractions (which are mathematically the difference of two large numbers) are small over most of the fault surface, and due to the inherent discontinuities from the discretization of the fault surface (see upper-right and lower-right panels of Figure 4-2 and Figure 4-3). It was checked that such fluctuations did not exist for a planar fault surface, which yielded zero residuals everywhere on the fault surface from Equation 1. These fluctuations for the 3D fault geometry can, however, be minimized if curvature-based adaptive meshing is used everywhere on the fault surface, but the current limit on the maximum mesh size did not allow the use of meshes having high resolutions far from the asperities. Once kernels have been computed, they are stored and used for multiple simulation scenarios (or "runs"). In order to check model inputs going into the Matlab forward solver, *EvolveSlip*, another "dashboard" of model inputs is automatically generated for Paraview after generating all inputs for a run (see Figure 4-4 for dip-slip motion and Figure 4-5 for strike-slip motion), including (a) the tractions on the fault surface due to slip on fault and

asperity patches, (b) surface displacement field due to unit asperity slip for every asperity, (c) rheological parameter distribution, and (d) the initial slip, or slip all over the fault surface immediately after each asperity ruptures.

To be more realistic, a narrow slip transition zone is included around each asperity over which coseismic slip drops to zero (see bottom right panel of Figure 4-4 and Figure 4-5). The effect of different parameterizations for the slip distribution in this transition zone (or slip-"tapering") are discussed in Hetland et al. [2010]. Several tapering functions are available in *Fslip.ModelInp* module for estimating this transitional slip. Typically, a fraction (0.25–0.5) of the asperity dimension is choosen to be the (arbitrary) width of this transition zone. As noted above, during meshing, this transition zone is where much of the refinement occurs to resolve coseismic tractions, and routine checks are carried out for each mesh to make sure that most of the coseismic stress pulse lies within this well-resolved transition zone, as far as practically possible.

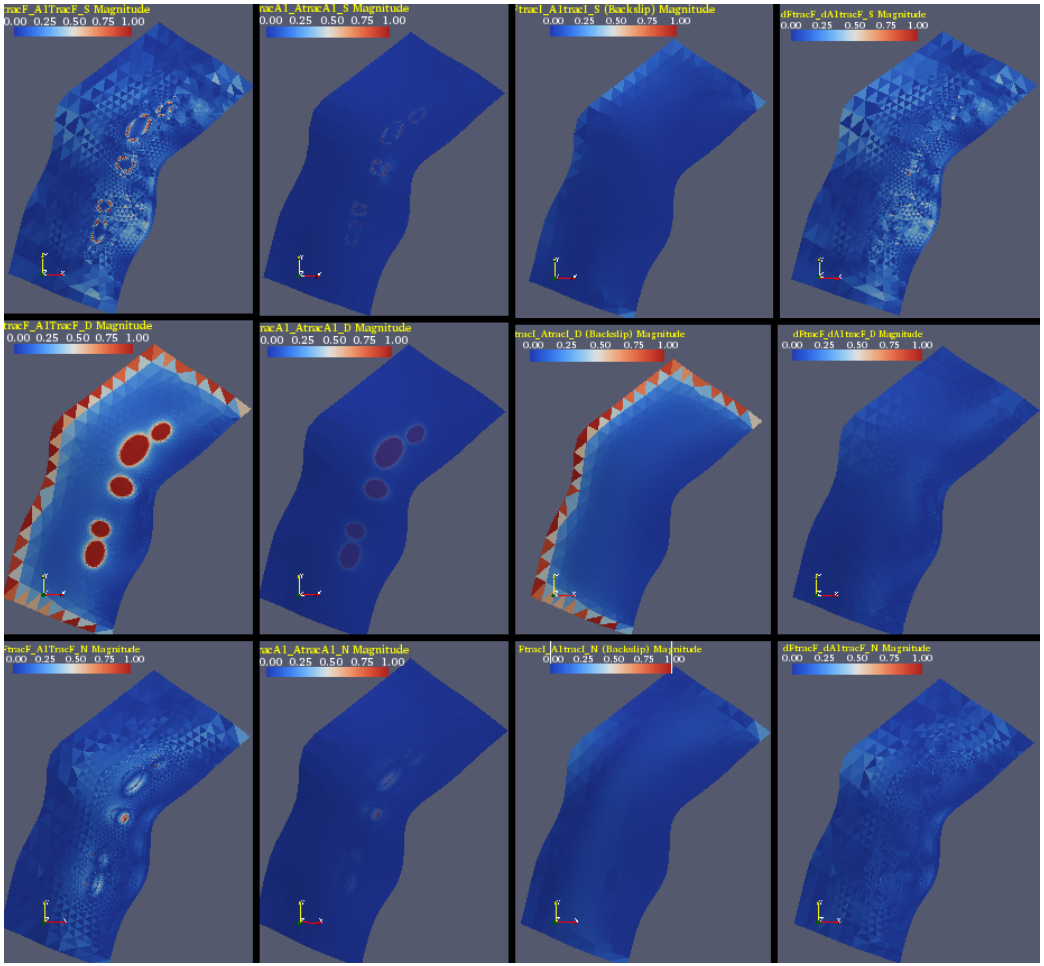


Figure 4-2. An automatically generated Paraview dashboard showing fault traction components (τ_S, top row; τ_D, middle row; τ_N, bottom row) due to dip-slip on all fault patches (column 1), slip on each asperity (column 2: the duller color is due to the use of transparency in Paraview to show all asperity contributions in a single panel), slip on the semi-infinite extensions of the fault (or backslip, column 3), and the residuals obtained from the superposition (Equation 1). For a planar fault, these residuals were found to be zero, as expected.

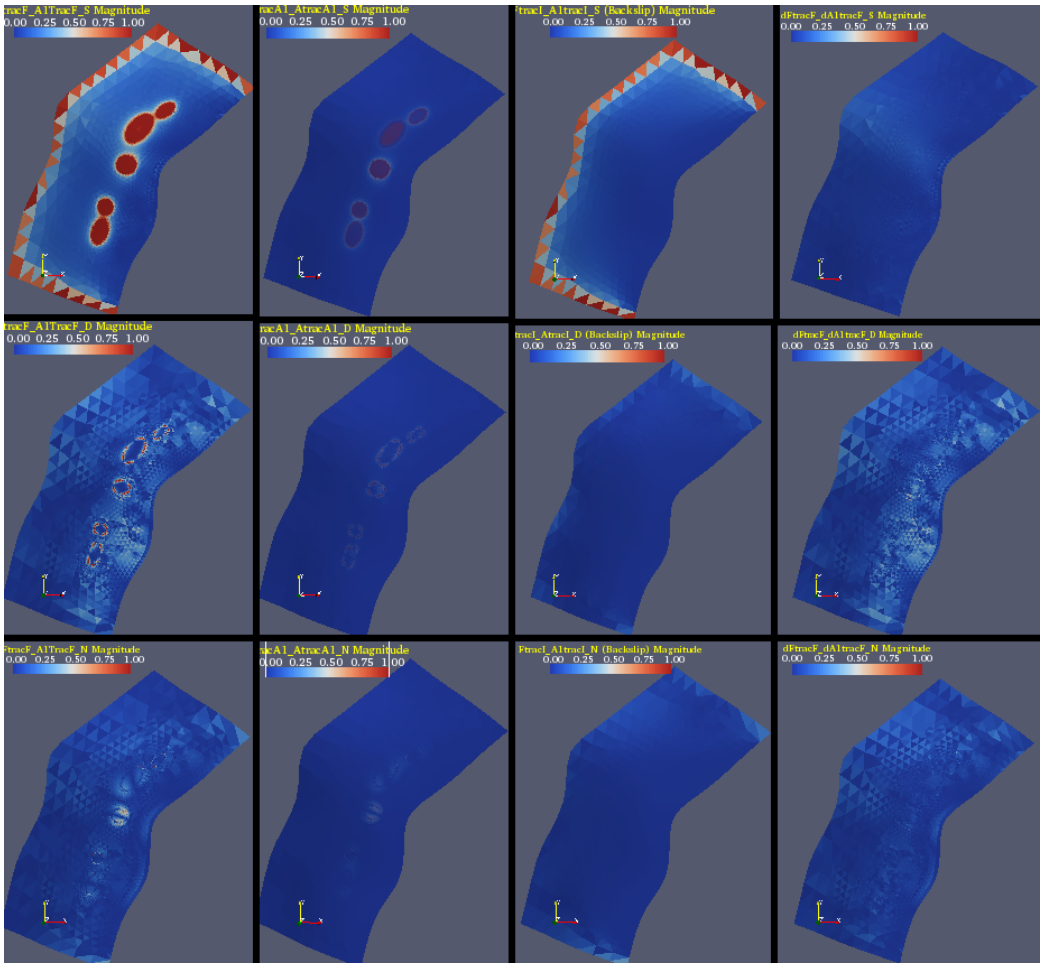


Figure 4-3. An automatically generated Paraview dashboard showing fault traction components (τ_S, top row; τ_D, middle row; τ_N, bottom row) due to strike-slip on all fault patches (column 1), slip on each asperity (column 2: again, the duller color is due to the use of transparency in Paraview to show all asperity contributions in a single panel), slip on the semi-infinite extensions of the fault (or backslip, column 3), and the residuals obtained from the superposition (Equation 1). For a planar fault, these residuals are zero, as expected.

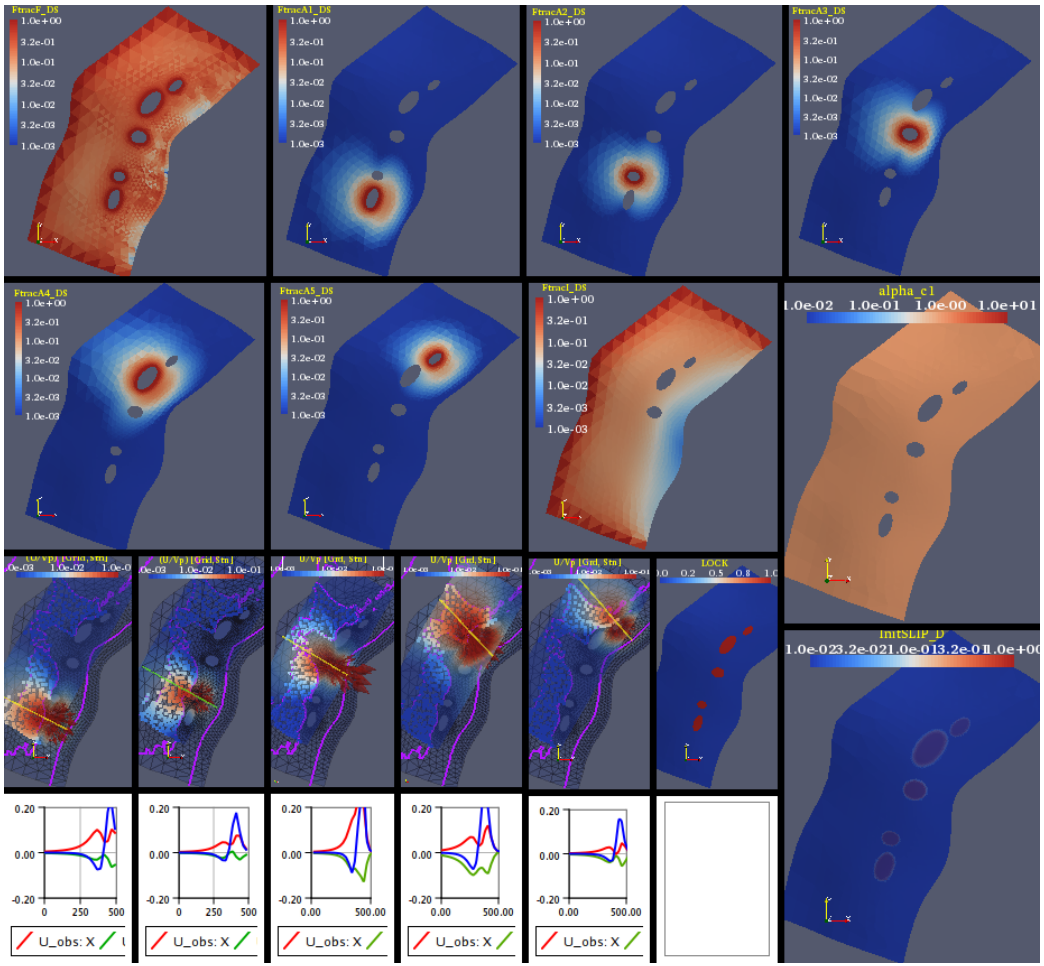


Figure 4-4. An automatically generated Paraview quality control dashboard for unit dip-slip on the fault and asperities, showing the rheological parameter distribution (rightmost middle panel), initial slip distribution (bottom-right panel), tractions (top two rows of panels), and surface displacements. Such figures are automatically generated and used for visualizing model inputs before a run or for debugging after a run. All 3D plots are colored on a log-scale, to bring out any small-scale heterogeneities.

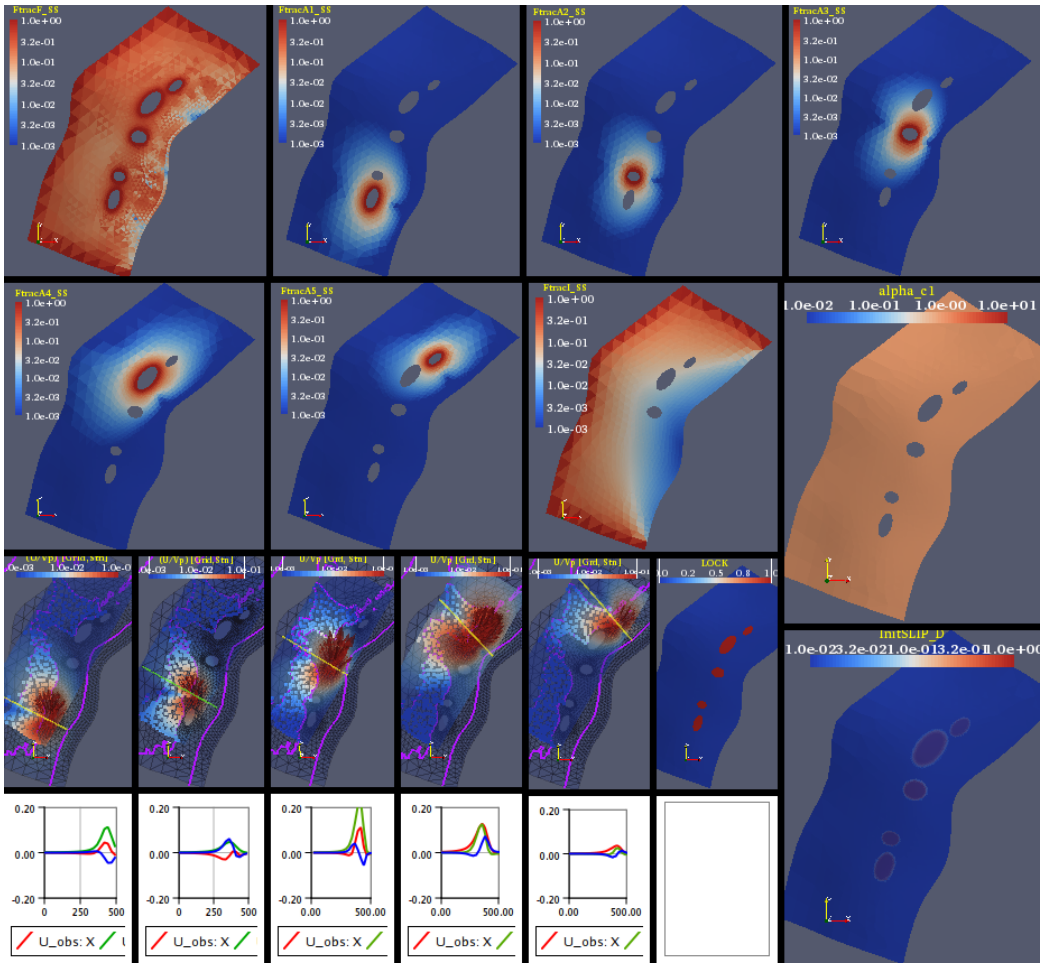


Figure 4-5. An automatically generated Paraview quality control dashboard for unit strike-slip on the fault and asperities, showing the rheological parameter distribution (rightmost middle panel), initial slip distribution (bottom-right panel), tractions (top two rows of panels), and surface displacements. Such figures are automatically generated and used for visualizing model inputs before a run or for debugging after a run. All 3D plots are colored on a log-scale, to bring out any small-scale heterogeneities.

4.5 Rheology

The Matlab forward slover, *EvolveSlip*, models linear-viscous, non-linear viscous, purely rate-dependent and Dietrich-Ruina rate-state rheologies. Hetland et al. [2010] showed that late in the seismic cycle, which is what we are attempting to simulate here, both purely rate-dependent as well as rate-state rheologies give very similar fault slip rates. In our simulations presented here, we use only purely rate-dependent rheology. However, for the purposes of intuition building in terms of the effects of strong vs. weak faults, we use a linear viscous rheology. For linear viscous rheology, slip-rate depends linearly on fault loading tractions, as opposed to the exponential dependence in typical rate-dependent rheologies. Therefore, simulations with linear viscous rheology can be used as a starting point in understanding the "strength" of the fault required to match surface velocity predictions to the observed velocities. While *EvolveSlip* can theoretically handle heterogeneous fault zone rheology, in this initial work, we only explore constant rheological properties, ignoring any lateral or depth dependence. In order to compare runs for different rheologies, all simulations are carried out with non-dimensional parameterizations. Hetland et al. [2010] cast the relationship between slip-rate and fault tractions for various rheologies in the form:

$$\dot{s} = f(\tau, \alpha) \tag{2}$$

where, α is a strength parameter that depends on the rheology.

For linear viscous rheology,

$$\dot{s} = f(\tau, \alpha) = \frac{\tau}{\alpha} \tag{3}$$

and α depends only on the characteristic viscosity, η, and width, h, of the fault zone:

$$\alpha = \frac{\eta}{h} \ . \tag{4}$$

The viscosity can be computed as the product of the shear-modulus, μ, and a relaxation time (in the case of Maxwell viscoelasticity), T_R. The non-dimensional scaling for the strength parameter is simply,

$$\alpha_0 = \frac{\tau_0}{V_0} = \frac{\mu S_0}{V_0 D_0} \tag{5}$$

where, V_0 is the loading rate (plate velocity), S_0 is the characteristic slip on the asperity, and D_0 is the characteristic asperity dimension. So, from (4) and (5), a dimensionless strength parameter for viscous rheology is:

$$\alpha' = \frac{\alpha}{\alpha_0} = \frac{\mu T_R}{h}\frac{V_0 D_0}{\mu S_0} = \frac{V_0 T_R}{S_0}\frac{D_0}{h} \ . \tag{6}$$

Non-dimensionalizing (3) using α_0, τ_0, and V_0, and rearranging, we obtain,

$$\tau' = \frac{\tau}{\tau_0} = \frac{\alpha \dot{s}}{\alpha_0 V_0} = \alpha' V' \ . \tag{7}$$

Thus, late in the cycle, when most of the fault is slipping at the loading rate, the mean dimensionless shear tractions along the fault surface will equal the dimensionless strength parameter, α'. Typical values for the above parameters are: $V_0 \approx 10^{-2}$ m/yr, $S_0 \approx 1$ m, $D_0 \approx 10^4$ m, h ≈ 1 m, and $T_R \approx 10^{-3}$–10^{-1} yr. Substituting these into (6) yields the range of values for α' as 0.1 to 10.

For rate-dependent rheology,

$$\dot{s} = f(\tau, \alpha) = V_0 e^{\left(\frac{-f_0}{(a-b)}\right)} \sinh\left(\frac{\tau}{(a-b)\sigma_0}\right) \tag{8}$$

where, f_0 is the static friction coefficient, a and b are the coefficients of the direct and indirect (state-dependent) dynamic frictional effects, and σ_0 is the effective normal traction on the fault surface. In our model, σ_0 can be both time-dependent (e.g., in response to slip evolution on the fault surface), as well as spatially heterogeneous (e.g., due to pore-pressure variations). In the demonstrative results presented in Chapter 5, we choose a constant σ_0 over the entire fault surface. The implications of such an assumption are discussed in more detain in Section 5.2. Therefore, there are two non-dimensional rheological parameters in this case [Hetland et al., 2010]:

$$V' = \frac{\dot{s}}{V_0} = e^{(-\rho)} \sinh\left(\frac{\tau'}{\alpha'}\right) \tag{9}$$

where, $\rho = f_0/(a-b)$, and $\alpha = (a-b)\sigma_0$. Thus, late in the cycle, when most of the fault is slipping at the loading rate, the mean dimensionless shear tractions along the fault surface

will equal ρ times the dimensionless strength parameter, α'. Typical values of $(a-b) \approx 10^{-2}$ [Blanpied et al., 1991; Marone et al., 1991], $f_0 \approx 0.1–1$, $\sigma_0 \approx 10–10^2$ MPa [Rice, 1993; Lapusta and Rice, 2003]. Thus, typical range of values for $\rho \approx 10–100$, and $\alpha \approx 10^5–10^6$ Pa. Assuming the same values as before for the non-dimensionalization of stresses, and $\mu \approx 10^{10}$ Pa, $\alpha' \approx 10^{-2}–10^{-1}$.

4.6 Significant developments over the existing Matlab forward solver, *EvolveSlip*

In addition to the general codes discussed above, several developments needed to be made to the existing Matlab forward solver, *EvolveSlip*, and its dependencies, in order to handle models with real fault surfaces. Here, we discuss the most significant changes. As before, we refer to Hetland et al. [2010], and Hetland and Simons [2010] as HSD10, and HS10, respectively.

Size of the problem: In HSD10 and HS10, a set rectangular region surrounding the asperities is always discretized at a fine resolution. This fine rectangular mesh around the asperities transitions into coarser rectangles towards the edges of the fault. In their implementation, asperities are defined at the time of running the model by designating patches lying within a specified region of the planar fault to be locked ("asperity"). Here, we use a different approach, whereby the asperities are defined at the time of meshing the megathrust interface, and fault parameter evolution is simulated only over the region of the fault outside these asperities. Driving coseismic stresses (and hence, the largest gradients in the modeled field parameters) occur along a narrow band surrounding the asperities. Therefore, as discussed in the meshing section, a very fine mesh resolution is applied over a narrow transition zone around the asperities. By not storing model parameter evolution over asperities during forward model calculations, the size of the Kernel array can be reduced by a factor of 40–50%, which can be significant when there are thousands of patches within the asperities. Also, a much higher patch size contrast is achieved between the fault edges and asperity boundaries, because a large fraction of the finer mesh residing within the asperities is ignored. In addition, the above references did

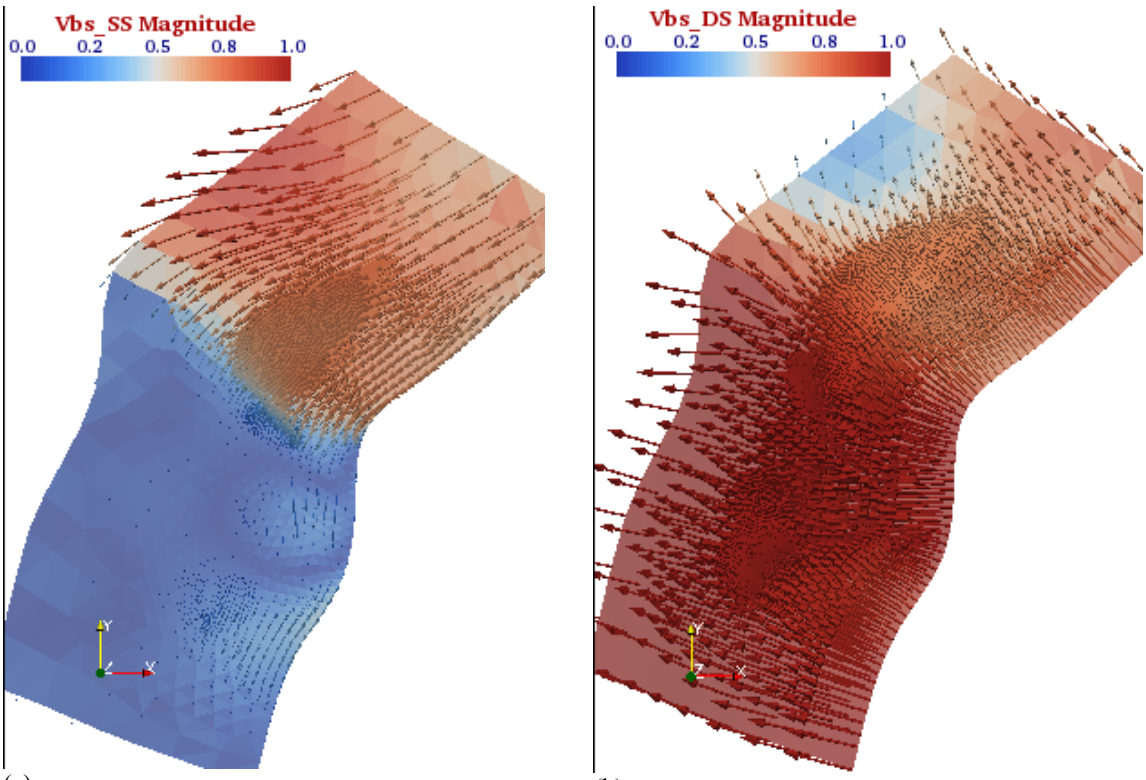
not consider surface deformation at observation stations (which can number in the several 100s to 1000, for a place like Japan). So, excluding the evolution of model parameters over asperity surfaces allows us to not only resolve the driving tractions after each rupture at a much finer scale than in the scheme implemented in Hetland et al. (by roughly an order of magnitude), but also to include thousands of surface observation points (actual stations as well as a surface observation grid for visualizing surface deformation).

Rupture sequence and timing of rupture: Since HSD10 and HS10 were trying to demonstrate a method for simulating postseismic fault slip, they did not consider realistic rupture sequences. In reality, the rupture interval for each asperity may not be related to that of another asperity by whole fractions (because rupture timing offset, or rupture intervals, or both). In this case, the concept of a characteristic rupture sequence time (or CRS cycle) — which is arithmetically equivalent to the least common multiple of all asperity rupture intervals — is utilized. Essentially, after each CRS cycle, ruptures on all asperities repeat in the same sequence, and rupture-time offsets as the previous CRS cycle. Therefore, a different convergence criterion had to be developed for the case of an arbitrary sequence of ruptures on multiple asperities. The convergence criterion used in Hetland et al. was designed for relatively uniform meshes with small gradients in mesh size (arithmetic mean of tractions measured at a single time — just before a subsequent rupture). When using unstructured grids having a large range of mesh element sizes (as is the case here), it is more appropriate to use an area-weighted mean for checking convergence. In addition, when using realistic rupture sequences, the mean tractions vary significantly between the individual constituent earthquakes in each CRS-cycle, thus making it inappropriate to use only a single point in time to measure convergence. Here, we use moving averages taken over different time-spans to determine an appropriate time-scale over which the model is deemed to have converged. Not surprisingly, the smallest moving average window beyond which convergence is stable turns out to be the CRS-cycle time. Therefore, the first complete CRS-cycle after convergence gives the simulated earthquake sequence for the given set of asperities. Synthetic surface velocities are extracted from this cycle for comparison to surface geodetic data.

Application of backslip with varying rake over an arbitrary fault surface in 3D: HSD10 and HS10 considered only planar faults. For a given relative plate velocity orientation, the rake (angle between slip vector and down-dip vector along the fault plane) is constant across the fault surface. However, for an arbitrary dipping fault surface in 3D, rake is a spatially varying quantity, whose local rate of variation depends on the degree of smoothing applied to that surface (as discussed in the meshing and kernels sections above). As originally postulated by Savage [1983], and demonstrated by Kanda and Simons [2010], backslip over a curved megathrust interface must be applied only along the local tangent to the fault surface. So, an additional feature was introduced whereby backslip is prescribed locally, over every patch depending on its orientation (strike and dip) relative to that of the plate velocity vector. Also, depending on the relation between the plate velocity vector direction and trench strike along the megathrust interface, it is possible to have strike-slip component of the plate velocity having both "up-strike" and "down-strike" directions along the same mega-thrust interface. In order to handle both positive and negative slip-rates in a given slip-direction, Kernel pre-multipliers computed from the local backslip unit vector were used to determine their correct response. Using a local backslip distribution results in significant strike-parallel fault slip partitioning (Figure 4-6) along the northern Japan-Kurile trench, with negligible along-strike slip for the southern Japan-Trench. This is consistent with slip inferred from observed surface geodetic measurements [see for example, DeMets, 1992; Loveless and Meade, 2010].

4.7 Steps in running a model

All model parameters (numbering about 150 for a typical run) are input into a comprehensively documented Input file, *ModelData.py*, consisting of three python data-classes — *Global Data*, *Kernel Data*, and *Run Data*. Given the number of inputs, and for



(a) (b)

Figure 4-6. (a) Strike-slip, and (b) dip-slip components of the non-dimensional backslip velocity field applied to the 3D fault surface. Color-scale indicates the same range of magnitudes in both figures as well as for the arrows.

tight quality control, this same input file is read by all the python drivers in *Fslip*, as well as the Matlab driver for *EvolveSlip*. A parser/translator was written in Matlab (and extensively tested) that can understand simple python assignment statements, including: lists, tuples, 1D and 2D arrays, concatenated strings, intrinsic functions and simple arithmetic expressions.

Currently, there are three python driver scripts and one matlab driver script for carrying out the different components of a simulation: (1) *Mesh2Kernels.py*: Mesh data extraction and kernel generation (typically, a one-time process for a given mesh resolution), (2) *ModelPar.py*: Python pre-processing run to generate model input parameters (Coseismic slip components, rupture history, Initial slip and rheological parameters distributions, locked fault patches — for every run), (3) *RunPyModel.m*: Forward model calculations,

using a Matlab driver that parses the input file, *ModelData.py*, extracts the kernels as well as model parameters from binary files, assembles the kernel matrix, sets additional run-time parameters, and finally, runs *EvolveSlip*, and (4) *Matbin2Vtk.py*: Post-procesing to probe and extract binary Matlab output for further processing as well as VTK files for visualization.

In order to aid in debugging and check for any run-time issues, the drivers currently output a large amount of screen output to indicate different stages of activity. In the future a verbosity flag will be added to suppress this output when not required. Also, as part of validating the code, we tested a series of runs for both spin-up and consistency of surface velocity predictions. Different fault-asperity configurations were used along with either linear viscous (for benchmarking) or rate strengthening rheology: (a) single asperity on a planar fault approximating the Tohoku section of the Japan Trench (TL), (b) entire 3D fault along the Japan Trench, with one (JT1) and two (JT2) asperities, (c) entire 3D fault along the Japan Trench, with the five major asperities (discussed in the next chapter), both without (JT5) and with variable backslip rake (JT5vb). Due to space constraints only the results from the last run (JT5vb) are presented in the next chapter.

4.8 Future directions for research and development

More sophisticated scientific questions can be addressed with extensions to the software developed here, for instance:

- The effect of lateral and depth variations in fault rheology.
- As the spatio-temporal resolution of geodetic data gets better, and crustal structure gets better resolved using seismic data, we could also use kernels generated accounting for local 3D crustal heterogeneities. Such 3D kernels would have to be computed numerically, using for example, a Finite Element Method (FEM) code, but are straightforward to compute at the present time.
- A long-term challenge is to attain the ability to simultaneously model multiple rheologies on the fault surface, which result in slip evolution over multiple length and

time-scales on the fault surface. This will involve a thorough re-formulation of the meshing schemes, as well as the forward model solver, taking into account any potential numerical instabilities resulting from such multi-scale evolution.

Thus, future software development can proceed in several directions. First, we want to migrate all the forward modeling functionality (currently in the Matlab scripts *EvolveSlip*) to python in order to avoid the array size limitations imposed by Matlab — so higher resolution discretizations and/or larger fault areas can be handled.
The current code can also be speeded by simple changes such as eliminating verbose screen output and further optimization of file-I/O. However, in order to make it feasible to routinely carry out inversions of geodetic data for fault rheological properties, significant speed improvements are needed. Such improvements are possible by recognizing that the elastic fields due to slip on a fault patch have an inherent length-scale (roughly a few times the characteristic patch size). So, at the present, the kernel matrix is very "sparse" in a numerical sense. Therefore, by introducing scale-dependent kernels, a kernel matrix with a much smaller band-width can be used for the forward model. However, given that we are testing non-linear rheologies, the inherent time-stepping (even if adaptive) limits the ultimate speed improvement that is possible for this problem, irrespective of the numerical integration scheme used. However, even now, simulations can be run and processed within a couple of days using an "embarrassingly-parallel" approach — that is, by simultaneously launching jobs on thousands of processors. In this sense, even now, a limited Bayesian type inversion is currently feasible within a reasonable time-frame. However, with increasing computing power (e.g., use of Graphical Processing Units, or GPUs) and cluster sizes, inverting for fault rheogical properties in a fully Bayesian sense will become even more tractable in the coming years.

References

Blanpied, M., D. Lockner and D. Byerlee (1991), Fault stability at hydrothermal conditions, Geophys. Res. Lett., 18**,** 609–612.

DeMets, C. (1992), Oblique Convergence and Deformation Along the Kuril and Japan Trenches, J. Geophys. Res., 97**,** 17,615–617,625, doi:610.1029/1092JB01306.

Gocad, 2010. (http://www.gocad.org/www/gocad/index.xhtml) Gocad Research Group.

Henderson, A. (2004), The Paraview Guide, 1st edn, vol., Kitware Inc.

Hetland, E.A. and M. Simons (2010), Postseismic and interseismic deformation due to fault creep II: Transient creep and interseismic stress shadows on megathrusts. , Geophys. J. Int., 181**,** 99–112, doi:110.1111/j.1365-1246X.2009.04482.x.

Hetland, E.A., M. Simons and E.M. Dunham (2010), Postseismic and interseismic deformation due to fault creep I: Model description. , Geophys. J. Int., 181**,** 81–98, doi:10.1111/j.1365-1246X.2010.04522.x.

Ito, A., G. Fujie, T. Tsuru, S. Kodaira, A. Nakanishi and Y. Kaneda (2004), Fault plane geometry in the source region of the 1994 Sanriku-oki earthquake Earth Planet. Sci. Lett., 223**,** 163-175.

Iwasaki, T., W. Kato, T. Moriya, A. Hasemi, N. Umino, T. Okada, K. Miyashita, T. Mizogami, T. Takeda, S. Seikine, T. Matsushima, K. Tashiro and H. Miyamachi (2001), Extensional structure in northern Honshu arc as inferred from seismic refraction/wide-angle reflection profiling, Geophys. Res. Lett., 28**,** 2329–2332.

Kanda, R.V.S. and M. Simons (2010), An elastic plate model for interseismic deformation in subduction zones, J. Geophys. Res., 115**,** B03405.

Lapusta, N. and J. Rice (2003), Nucleation and early seismic propagation of small and large events in a crustal earthquake model, J. geophys. Res., 108**,** 2205, doi:2210.1029/2001JB000793.

Loveless, J.P. and B.J. Meade (2010), Geodetic imaging of plate motions, slip rates, and partitioning of deformation in Japan, J. Geophys. Res., 115**,** B02410, doi:02410.01029/02008JB006248.

Marone, C., C. Scholz and R. Bilham (1991), On the mechanics of earthquake afterslip, J. geophys. Res., 96**,** 8441–8452.

Meade, B.J. (2007), Algorithms for the calculation of exact displacements, strains, and stresses for triangular dislocation elements in a uniform elastic half space. , Comp. Geosci., 33**,** 1064-1075, doi:1010.1016/j.cageo.2006.1012.1003.

Miura, S., S. Kodaira, A. Nakanishi and T. Tsuru (2003), Structural characteristics controlling the seismicity of southern Japan Trench fore- arc region, revealed by ocean bottom seismographic data, Tectonophysics, 363**,** 79-102.

Miura, S., N. Takahashi, A. Nakanishi, T. Tsuru, S. Kodaira and Y. Kaneda (2005), Structural characteristics off Miyagi forearc region, the Japan Trench seismogenic zone, deduced from a wide-angle reflection and refraction study, Tectonophysics, 407**,** 165-188.

Nakanishi, A., A.J. Smith, S. Miura, T. Tsuru, S. Kodaira, K. Obana, N. Takahashi, P.R. Cummins and Y. Kaneda (2004), Structural factors controlling the coseismic rupture zone of the 1973 Nemuro-Oki earthquake, the southern Kuril Trench seismogenic zone, J. Geophys. Res., 109**,** B05305, doi: 05310.01029/02003JB002574.

Okada, Y. (1992), Internal deformation due to shear and tensile faults in a half-space, Bull. Seismol. Soc. Am., 82**,** 1018-1040.

Owen, S.J., 2006. CUBIT 10.2 Documentation, pp. 532, Sandia National Laboratories, Albuquerque, NM, U.S.A.

Rice, J. (1993), Spatio-temporal complexity of slip on a fault, J. geophys. Res. , 98**,** 9885–9907.

Savage, J.C. (1983), A dislocation model of strain accumulation and release at a subduction zone, J. Geophys. Res., 88 4984-4996.

Schroeder, W., K. Martin and B. Lorensen (2006), The Visualization Toolkit An Object-Oriented Approach To 3D Graphics, 4th edn, vol., Kitware Inc. .

Takahashi, N., S. Kodaira, T. Tsuru, J.-O. Park, Y. Kaneda, K. Suyehiro, H. Kinoshita, S. Abe, M. Nishino and R. Hino (2004), Seismic structure and seismogenesis off Sanriku region, northeastern Japan, Geophys. J. Int., 159**,** 129-145.