Crustal Deformation Modeling Tutorial

PyLith and CUBIT/Trelis Refresher

Brad Aagaard
Charles Williams
Matthew Knepley

COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

June 23, 2014

Crustal Deformation Modeling

Elasticity problems where geometry does not change significantly

Quasi-static modeling associated with earthquakes

- Strain accumulation associated with interseismic deformation
  - What is the stressing rate on faults X and Y?
  - Where is strain accumulating in the crust?
- Coseismic stress changes and fault slip
  - What was the slip distribution in earthquake A?
  - How did earthquake A change the stresses on faults X and Y?
- Postseismic relaxation of the crust
  - What rheology is consistent with observed postseismic deformation?
  - Can aseismic creep or afterslip explain the deformation?

Crustal Deformation Modeling

Elasticity problems where geometry does not change significantly

Dynamic modeling associated with earthquakes

- Modeling of strong ground motions
  - Forecasting the amplitude and spatial variation in ground motion for scenario earthquakes
- Coseismic stress changes and fault slip
  - How did earthquake A change the stresses on faults X and Y?
- Earthquake rupture behavior
  - What fault constitutive models/parameters are consistent with the observed rupture propagation in earthquake A?

Crustal Deformation Modeling

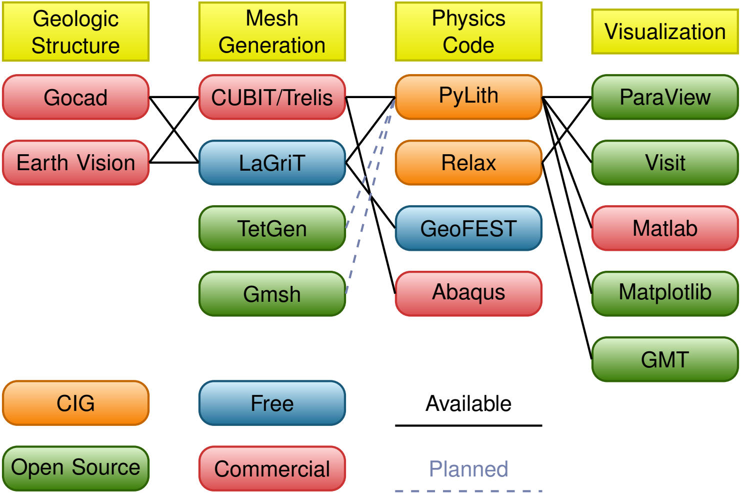
Elasticity problems where geometry does not change significantly

Volcanic deformation associated with magma chambers and/or dikes

- Inflation
  - What is the geometry of the magma chamber?
  - What is the potential for an eruption?
- Eruption
  - Where is the deformation occurring?
  - What is the ongoing potential for an eruption?
- Dike intrusions
  - What is the geometry of the intrusion?
  - What is the pressure change and/or amount of opening/dilatation?

Crustal Deformation Modeling

Overview of workflow for typical research problem



PyLith

- Developers
  - Brad Aagaard (USGS, lead developer))
  - Charles Williams (GNS Science, formerly at RPI)
  - Matthew Knepley (Univ. of Chicago, formerly at ANL)
- Combined dynamic modeling capabilities of EqSim (Aagaard) with the quasi-static modeling capabilities of Tecton (Williams)
- Use modern software engineering (modular design, testing, documentation, distribution) to develop an open-source, community code

Governing Equations

Elasticity equation

$$\sigma_{ij,j} + f_i = \rho\ddot{u} \text{ in } V, \tag{1}$$

$$\sigma_{ij}n_j = T_i \text{ on } S_T, \tag{2}$$

$$u_i = u_i^0 \text{ on } S_u, \text{ and} \tag{3}$$

$$R_{ki}(u_i^+ - u_i^-) = d_k \text{ on } S_f. \tag{4}$$

Multiply by weighting function and integrate over the volume,

$$-\int_V (\sigma_{ij,j} + f_i - \rho\ddot{u}_i)\phi_i \, dV = 0 \tag{5}$$

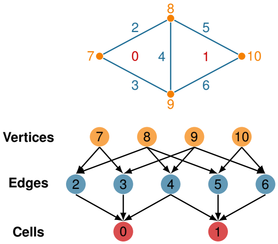
After some algebra,

$$-\int_V \sigma_{ij}\phi_{i,j} \, dV + \int_{S_T} T_i\phi_i \, dS + \int_V f_i\phi_i \, dV - \int_V \rho\ddot{u}_i\phi_i \, dV = 0 \tag{6}$$

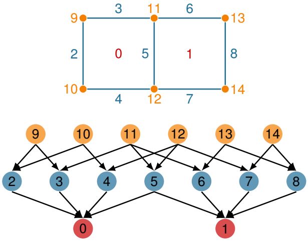
Discretize Domain Using Finite Elements

PyLith v2.0.0 uses interpolated meshes

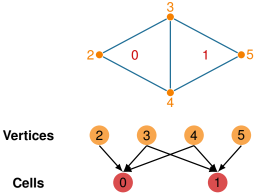
Interpolated triangular mesh



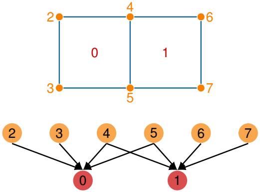
Interpolated quadrilateral mesh



Optimized triangular mesh



Optimized quadrilateral mesh



Governing Equations

Using numerical quadrature we convert the integrals to sums over the cells and quadrature points

$$
\begin{aligned}
-\sum_{\text{vol cells}} \sum_{\text{quad pts}} \sigma_{ij} N^n_{,j} w_q |J_{\text{cell}}| + \sum_{\text{surf cells}} \sum_{\text{quad pts}} T_i N^n w_q |J_{\text{cell}}| \\
+ \sum_{\text{vol cells}} \sum_{\text{quad pts}} f_i N^n w_q |J_{\text{cell}}| \\
- \sum_{\text{vol cells}} \sum_{\text{quad pts}} \rho \sum_m \ddot{a}^m_i N^m N^n w_q |J_{\text{cell}}| = \vec{0} \quad (7)
\end{aligned}
$$

Quasi-static Solution

Neglect inertial terms

Form system of algebraic equations

$$\underline{A}(t)\vec{u}(t) = \vec{b}(t) \tag{8}$$

where

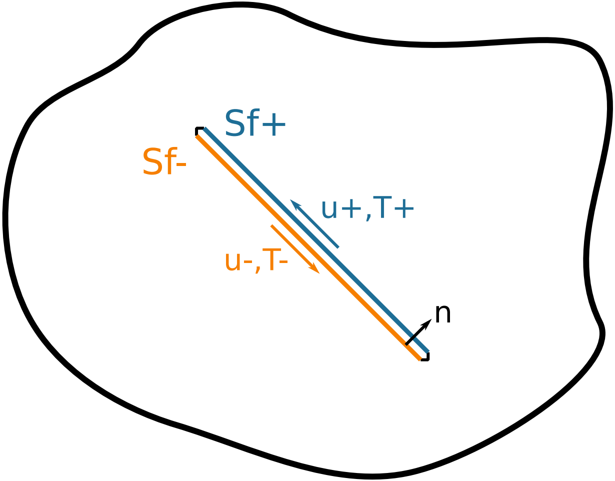
$$A_{ij}^{nm}(t) = \sum_{\text{vol cells}} \sum_{\text{quad pts}} \frac{1}{4} C_{ijkl}(t)(N_{,l}^{m} + N_{,k}^{m})(N_{,j}^{n} + N_{,i}^{n}) w_q |J_{\text{cell}}| \tag{9}$$

$$b_i(t) = \sum_{\text{surf cells}} \sum_{\text{quad pts}} T_i(t) N^n w_q |J_{\text{cell}}| + \sum_{\text{vol cells}} \sum_{\text{quad pts}} f_i(t) N^n w_q |J_{\text{cell}}| \tag{10}$$

and solve for $\vec{u}(t)$.

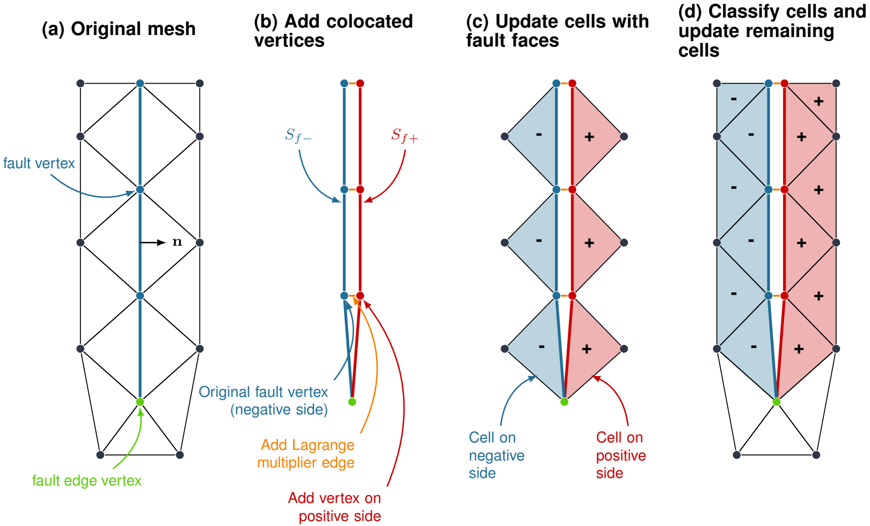
Fault Interface

Fault tractions couple deformation across interface



Implementation: Fault Interfaces

Use cohesive cells to control fault behavior



Fault Implementation: Governing Equations

Terms in governing equation associated with fault

- Tractions on fault surface are analogous to boundary tractions

$$\ldots + \underbrace{\int_{S_T} \vec{\phi} \cdot \vec{T}\, dS}_{\text{Neumann BC}} - \underbrace{\int_{S_{f+}} \vec{\phi} \cdot \vec{l}\, dS}_{\text{Fault +}} + \underbrace{\int_{S_{f-}} \vec{\phi} \cdot \vec{l}\, dS}_{\text{Fault -}} \ldots = 0$$

- Constraint equation relates slip to relative displacement

$$\int_{S_f} \vec{\phi} \cdot (\underbrace{\vec{d}}_{\text{Slip}} - \underbrace{(\vec{u}_+ - \vec{u}_-)}_{\text{Relative Disp.}}) dS = 0$$

Fault Slip Implementation

Use Lagrange multipliers to specify slip

- System without cohesive cells
  - Conventional finite-element elasticity formulation

$$\underline{A}\vec{u} = \vec{b}$$

  - Fault slip associated with relative displacements across fault

$$\underline{C}\vec{u} = \vec{d}$$

- System with Lagrange multiplier constraints for fault slip

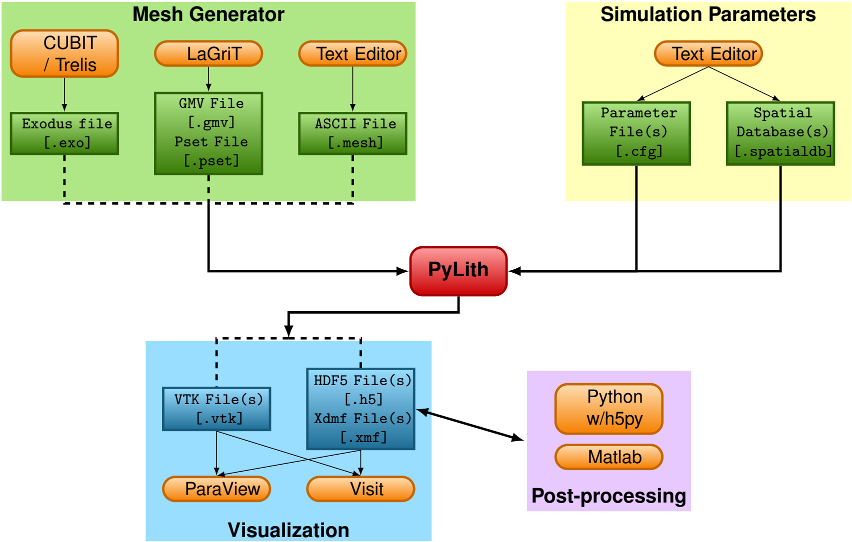
$$\begin{pmatrix} \underline{A} & \underline{C}^T \\ \underline{C} & 0 \end{pmatrix} \begin{pmatrix} \vec{u} \\ \vec{l} \end{pmatrix} = \begin{pmatrix} \vec{b} \\ \vec{d} \end{pmatrix}$$

- Prescribed (kinematic) slip
  Specify fault slip ($\vec{d}$) and solve for Lagrange multipliers ($\vec{l}$)
- Spontaneous (dynamic) slip
  Adjust fault slip to be compatible with fault constitutive model

Implementing Fault Slip with Lagrange multipliers

- Advantages
  - Fault implementation is local to cohesive cell
  - Solution includes tractions generating slip (Lagrange multipliers)
  - Retains block structure of matrix, including symmetry
  - Offsets in mesh mimic slip on natural faults
- Disadvantages
  - Cohesive cells require adjusting topology of finite-element mesh
  - Scalable preconditioner/solver is more complex

Workflow for Running PyLith



Spatial Databases

User-specified field/value in space

- Examples
  - Uniform value for Dirichlet (0-D)
  - Piecewise linear variation in tractions for Neumann BC (1-D)
  - SCEC CVM-H seismic velocity model (3-D)
- Generally independent of discretization for problem
- Available spatial databases

UniformDB	Optimized for uniform value
SimpleDB	Simple ASCII files (0-D, 1-D, 2-D, or 3-D)
SCECCVMH	SCEC CVM-H seismic velocity model v5.3
ZeroDispDB	Special case of UniformDB

Features in PyLith 2.0

Complete rewrite of finite-element data structures

- Time integration schemes and elasticity formulations
  - Implicit for quasistatic problems (neglect inertial terms)
    - Infinitesimal strains
    - Small strains
  - Explicit for dynamic problems
    - Infinitesimal strains
    - Small strains
    - Numerical damping via viscosity
- Bulk constitutive models (2-D, and 3-D)
  - Elastic model
  - Linear Maxwell viscoelastic models
  - Generalized Maxwell viscoelastic models
  - Power-law viscoelastic model
  - Drucker-Prager elastoplastic model

Features in PyLith 2.0 (cont.)

- Boundary and interface conditions
  - Time-dependent Dirichlet boundary conditions
  - Time-dependent Neumann (traction) boundary conditions
  - Absorbing boundary conditions
  - Kinematic (prescribed slip) fault interfaces w/multiple ruptures
  - Dynamic (friction) fault interfaces
  - Time-dependent point forces
  - Gravitational body forces
- Fault constitutive models
  - Static friction
  - Linear slip-weakening
  - Linear time-weakening
  - Dieterich-Ruina rate and state friction w/ageing law

Features in PyLith 2.0 (cont.)

- Automatic and user-controlled time stepping
- Ability to specify initial stress/strain state
- Importing meshes
  - LaGriT: GMV/Pset
  - CUBIT: Exodus II
  - ASCII: PyLith mesh ASCII format (intended for toy problems only)
- Output: VTK and HDF5 files
  - Solution over volume
  - Solution over surface boundary
  - Solution interpolated to user-specified points
  - State variables (e.g., stress and strain) for each material
  - Fault information (e.g., slip and tractions)
- Automatic conversion of units for all parameters
- Parallel uniform global refinement
- PETSc linear and nonlinear solvers
  - Custom preconditioner with algebraic multigrid solver

PyLith Development

See PyLith User Resources for detailed development plan

- Immediate priorities [in progress]
  - New fault implementation for spontaneous rupture
    Much faster convergence for quasi-static simulations
  - Improved handling of fault intersections
- Short-term priorities
  - Under-the-hood improvements
    - Support higher order basis functions [in progress]
      Provides much higher resolution for a given mesh
    - Multigrid nonlinear solver [in progress]
    - Prepare for multi-physics
  - Multi-cycle earthquake modeling
    Resolve interseismic, coseismic, and postseismic deformation
    - Coupling solvers for quasistatic and dynamic deformation
    - Adaptive time stepping
  - Multiphysics: Elasticity + Fluid flow + Heat flow
  - Scaling to 1000 cores

PyLith Development

Planned Releases

- v2.1 (Summer 2014)
  - New fault implementation for spontaneous rupture
  - Improved handling of fault intersections
- v3.0 (Early 2015)
  - Support for higher order basis functions
  - Adaptive time stepping
- v3.1 (Mid-Late 2015)
  - Support for incompressible elasticity
  - Heat and fluid flow coupled to elastic deformation

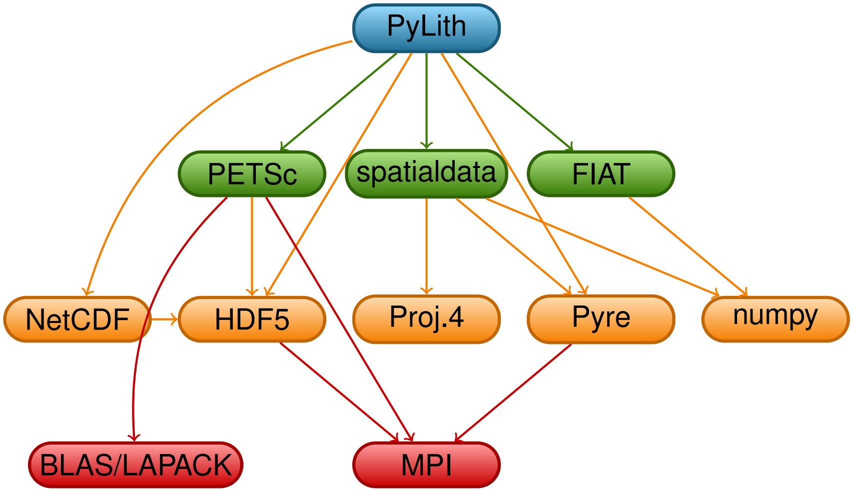
Design Philosophy

Modular, extensible, and smart

- Code should be flexible and modular
- Users should be able to add new features without modifying code, for example:
  - Boundary conditions
  - Bulk constitutive models
  - Fault constitutive models
- Input/output should be user-friendly
- Top-level code written in Python (expressive, dynamic typing)
- Low-level code written in C++ (modular, fast)

PyLith Design: Focus on Geodynamics

Leverage packages developed by computational scientists



COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

PyLith Application Flow

PyLithApp

```
main()
  mesher.create()
  problem.initialize()
  problem.run()
```

TimeDependent (Problem)

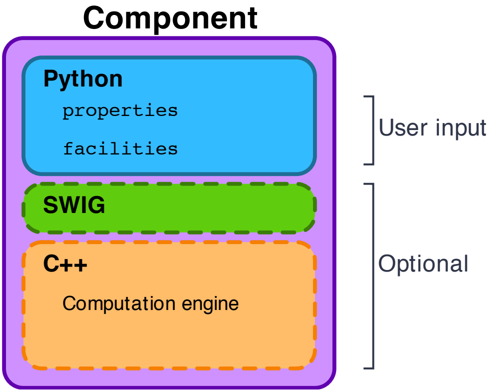
```
initialize()
  formulation.initialize()

run()
  while (t < tEnd)
    dt = formulation.dt()
    formulation.prestep(dt)
    formulation.step(dt)
    formulation.poststep(dt)
```

Implicit (Formulation)

```
initialize()

prestep()
  set values of constraints

step()
  compute residual
  solve for disp. incr.

poststep()
   update disp. field
   write output
```

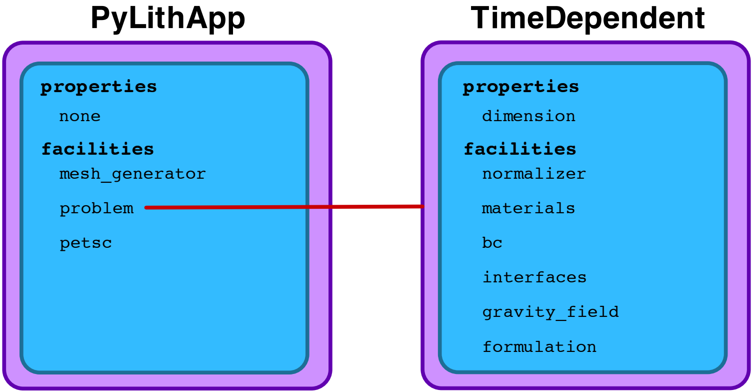
PyLith as a Hierarchy of Components

Components are the basic building blocks



PyLith as a Hierarchy of Components

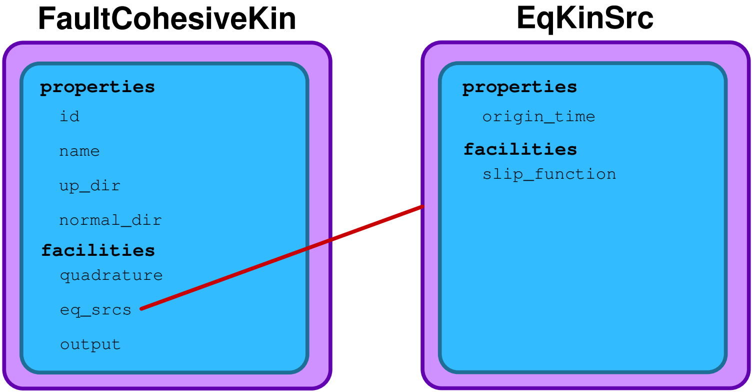
PyLith Application and Time-Dependent Problem



COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

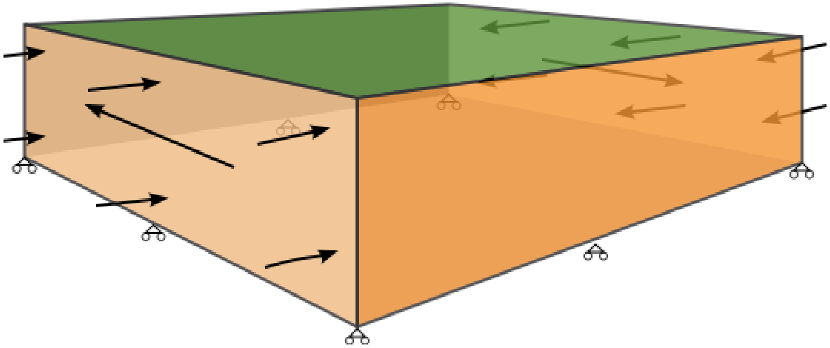
PyLith as a Hierarchy of Components

Fault with kinematic (prescribed slip) earthquake rupture

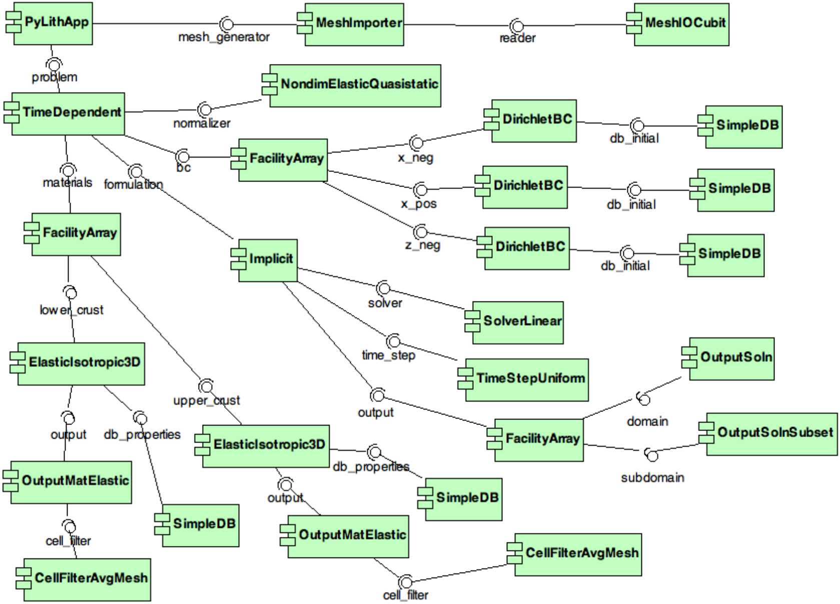


PyLith as a Hierarchy of Components

Diagram of simple toy problem



PyLith as a Hierarchy of Components



Unit and Regression Testing

Automatically run more than 1800 tests on multiple platforms whenever code is checked into the source repository.

- Create tests for nearly every function in code during development
  - Remove most bugs during initial implementation
  - Isolate and expose bugs at origin
- Create new tests to expose reported bugs
  - Prevent bugs from reoccurring
- Rerun tests whenever code is changed
  - Code continually improves (permits optimization with quality control)
- Binary packages generated automatically upon successful completion of tests
- Additional full-scale parallel regression tests are run before releases

Mesh Generation Tips

There is no silver bullet in finite-element mesh generation

- Hex/Quad versus Tet/Tri
  - Hex/Quad are slightly more accurate and faster
  - Tet/Tri easily handle complex geometry
  - Easy to vary discretization size with Tet, Tri, and Quad cells
  - There is no easy answer
    For a given accuracy, a finer resolution Tet mesh that varies the discretization size in a more optimal way *might* run faster than a Hex mesh
- Check and double-check your mesh
  - Were there any errors when running the mesher?
  - Are the boundaries, etc marked correctly for your BC?
  - Check mesh quality (aspect ratio should be close to 1)

CUBIT Workflow

1. Create geometry
   1. Construct surfaces from points, curves, etc or basic shapes
   2. Create domain and subdivide to create any interior surfaces
      - Fault surfaces must be interior surfaces (or a subset) that completely divide domain
      - Need separate volumes for different constitutive *models, not parameters*
2. Create finite-element mesh
   1. Specify meshing scheme
   2. Specify mesh sizing information
   3. Generate mesh
   4. Smooth to fix any poor quality cells
3. Create nodesets and blocks
   1. Create block for each constitutive model
   2. Create nodeset for each BC and fault
   3. Create nodeset for buried fault edges
   4. Create nodeset for ground surface for output (optional)
4. Export mesh in Exodus II format (`.exo` files)

CUBIT/Trelis Issues

Keep in mind the scales of the observations you are modeling

- Topography/bathymetry
  - Ignore topography/bathymetry unless you know it matters
  - For rectilinear grid, create UV net surface
  - Convert triangular facets to UV net surface via mapped mesh
- Fault surfaces
  - Building surfaces from contours is usually easiest
  - Include features at the resolution that matters
- Performance
  - Number of points in spline curves/surfaces has huge affect on mesh generation runtime
  - CUBIT/Trelis do not run in parallel
  - Use uniform global refinement in PyLith for large sims ($>$10M cells)

CUBIT/Trelis Best Practices

Issue: Changes in geometry cause changes in object ids
Soln: Name objects and use APREPRO or Python to eliminate hardwired ids wherever possible

Issue: Splines with many points slows down operations
Soln: Reduce the number of points per spline

Issue: Surfaces meet in small angles creating distorted cells
Soln: Trim geometry to eliminate features smaller than cell size

Issue: Difficulty meshing complex geometry with Hex cells
Soln: Use Tet cells even if it requires a finer mesh

Issue: Hex mesh over-samples parts of the domain
Soln: Use Tet mesh and vary discretization within domain

Issue: Extended surfaces create very complex geometry
Soln: Subdivide geometry before webcutting to eliminate overly complex geometry

General Numerical Modeling Tips

Start simple and progressively add complexity and increase resolution

- Start in 2-D, if possible, and then go to 3-D
  - Much smaller problems $\Rightarrow$ much faster turnaround
  - Start with an exact solver
  - Experiment with meshing, boundary conditions, solvers, etc
  - Keep in mind how physics differs from 3-D
- Start with coarse resolution and then increase resolution
  - Much smaller problems $\Rightarrow$ much faster turnaround
  - Start with an exact solver
  - Experiment with meshing, boundary conditions, solvers, etc.
  - Increase resolution until solution resolves features of interest
    - Resolution will depend on spatial scales in BC, initial conditions, deformation, and geologic structure
    - Is geometry of domain important? At what resolution?
    - Displacement field is integral of strains/stresses
    - Resolving stresses/strains requires fine resolution simulations
- Use your intuition and analogous solutions to check your results!

PyLith Tips

- Read the PyLith User Manual
- Do not ignore error messages and warnings!
- Use an example/benchmark as a starting point
- Quasi-static simulations
  - Start with a static simulation and then add time dependence
  - Check that the solution converges at every time step
- Dynamic simulations
  - Start with a static simulation
  - Shortest wavelength seismic waves control cell size
- CIG Short-Term Crustal Dynamics mailing list
  `cig-short@geodynamics.org`
- PyLith User Resources
  `http://wiki.geodynamics.org/software:pylith:start`

PyLith Debugging Tools

- `pylithinfo [--verbose] [PyLith args]`
  Dumps all parameters with their current values to text file
- Command line arguments
  - `--help`
  - `--help-components`
  - `--help-properties`
  - `--petsc.start_in_debugger` (run in xterm)
  - `--nodes=N` (to run on N processors on local machine)
- Journal info flags turn on writing progress

```
[pylithapp.journal.info]
timedependent = 1
```

  - Turns on/off info for each type of component independently
  - Examples turn on writing lots of info to stdout using journal flags

Getting Started

- Read the PyLith User Manual
- Work through the examples
  - Chapter 7 of the PyLith manual
  - Input files are provided with the PyLith binary
    `src/pylith-2.0.0/examples`
  - Input files are provided with the PyLith source tarball
    `src/examples`
- Modify an example to look like a problem of interest