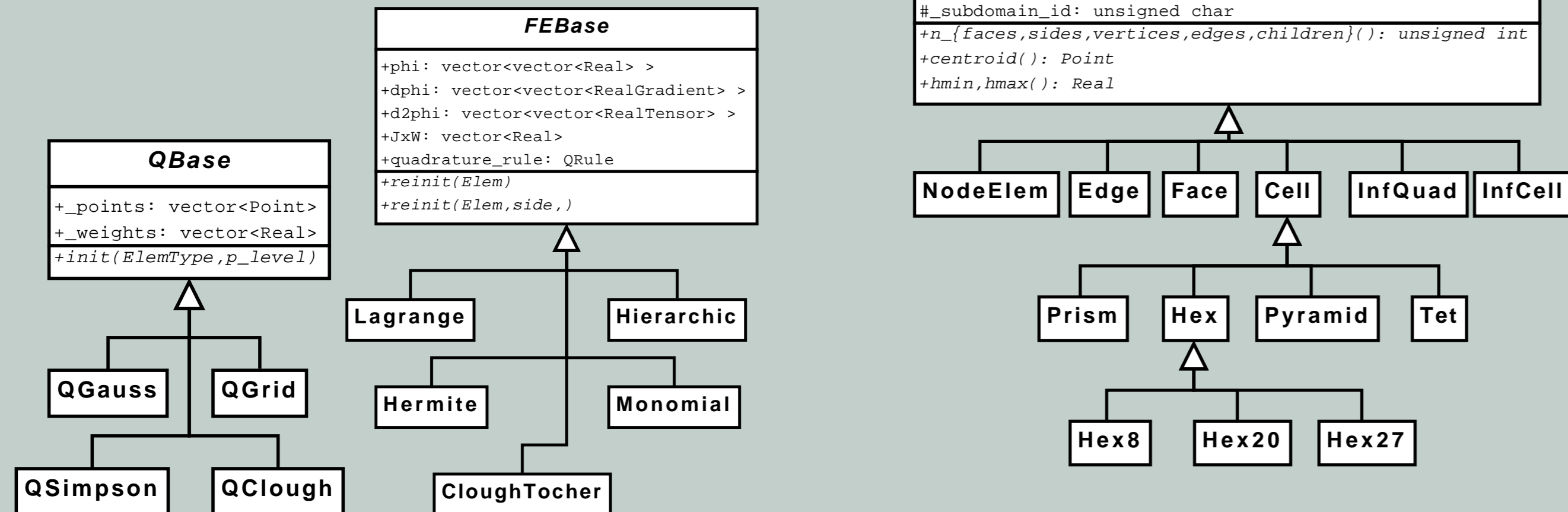


Core Features

- Support for mixed finite elements on hybrid unstructured grids in one through three dimensions.
- Assorted finite element classes including DG/FVM , C^0 , C^1 bases
- Adaptive h refinement with hanging nodes for arbitrary elements, p refinement for hierarchic bases
- Parallel system assembly and solution
- Parallel spatial adaptivity, projection, interpolation for general element types
- Nonlinear FEM framework with adaptive time integration
- Parallel distributed mesh data structures and algorithms
- Mesh and solution export and import using common data and visualization formats
- Integration with third party software packages such as:
 - PETSc, Trilinos, or LASPack sparse linear algebra
 - METIS, ParMETIS mesh partitioning
 - Triangle, Tetgen mesh generation
- Inline API documentation with Doxygen

Object Oriented Programming

- Applications are written to use Abstract Base Classes, e.g.:
 - Quadrature rule `QBase`
 - Geometric element `Elem`
 - Finite element class `FEBase`
- Then code is dispatched at runtime to Concrete Derived Classes, e.g.:
 - Gaussian quadrature `QGauss`
 - 3D higher order hexahedra `Hex27`
 - Hierarchic polynomials `FE<HIERARCHIC>`
- Optional higher level `FEMSystem`, `TimeSolver` class hierarchies encapsulate physics, time integration.



Verification

- **Regression Testing:** Automatic checking of example applications, unit tests via BuildBot
- **Assertions:** Debug-mode testing of internal state, application API usage
- GNU libstdc++ options verify container validity, index bounds-checking
- ~ 4000 high-level assertions added by developers

Manufactured Solutions:

$$\mathcal{R}_m(v) \equiv \mathcal{R}(v) - \mathcal{R}(u_m)$$

$$\frac{\partial \mathcal{R}_m}{\partial v} = \frac{\partial \mathcal{R}}{\partial v}$$

$$\mathcal{R}_m(u_m) = 0$$

Testing convergence of physics code with residual \mathcal{R} to arbitrary analytic solutions u_m via added forcing function

Convergence Rates:

$$e_h \propto \mathcal{O}(h^{p+1-r})$$

$$e_{\Delta t} \propto \mathcal{O}(\Delta t^{q+1})$$

$$e_{N,i} \propto \mathcal{O}(e_{N,i-1}^2)$$

Testing *a priori* convergence rates to known/unknown solutions on new applications

Jacobian Verification:

$$J \equiv \frac{\partial \mathcal{R}}{\partial u}$$

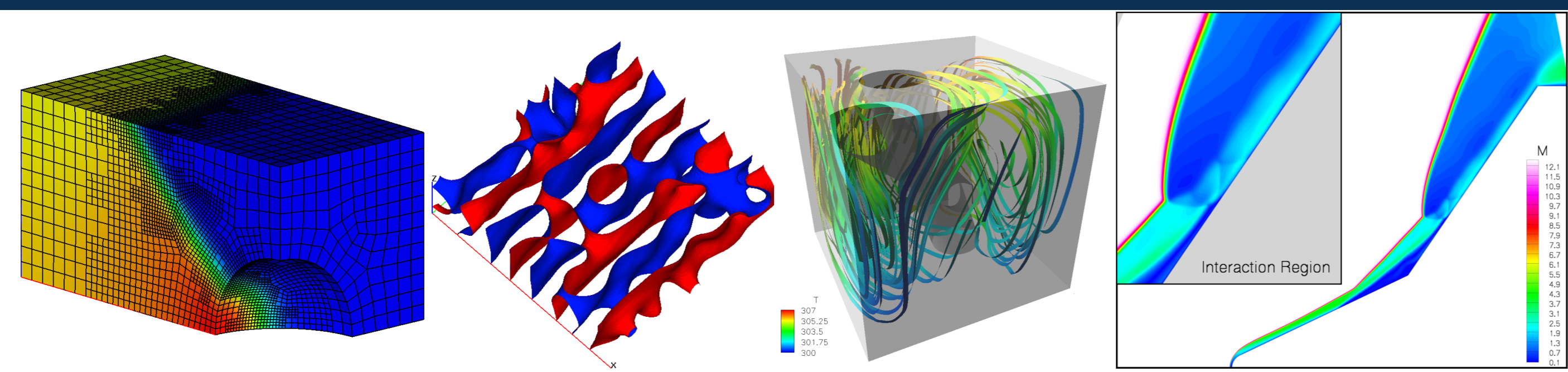
$$\tilde{J}_{ij} \equiv \frac{\mathcal{R}_i(u + \varepsilon e_j) - \mathcal{R}_i(u - \varepsilon e_j)}{2\varepsilon}$$

$$\|\tilde{J} - J\| < TOL \cdot \|\tilde{J}\|$$

Testing self-consistency of residual, analytic jacobian in application physics

Multiphysics

libMesh applications results have been published for compressible and incompressible flow and transport, phase change, angiogenesis, radiation transport, many other mathematical models.



Physics-Independent:

- No physics in library code (except example applications)
- APIs designed to facilitate natural translation from PDE—software

Multiphysics-Enabling:

- Support for arbitrary numbers of variables
 - Independent finite element types, orders
 - May vary between subdomains
 - Tightly or loosely coupled

$$\mathcal{R}(u_h, \phi_i) \equiv \int k(\vec{x}) \nabla u \cdot \nabla \phi_i d\Omega$$

$$\approx \sum_E \sum_q k(\vec{x}(\vec{\xi}_q)) \nabla u(\vec{x}(\vec{\xi}_q)) \cdot \nabla \phi_i(\vec{x}(\vec{\xi}_q)) \Big| J(\vec{\xi}_q) \Big| w_q$$

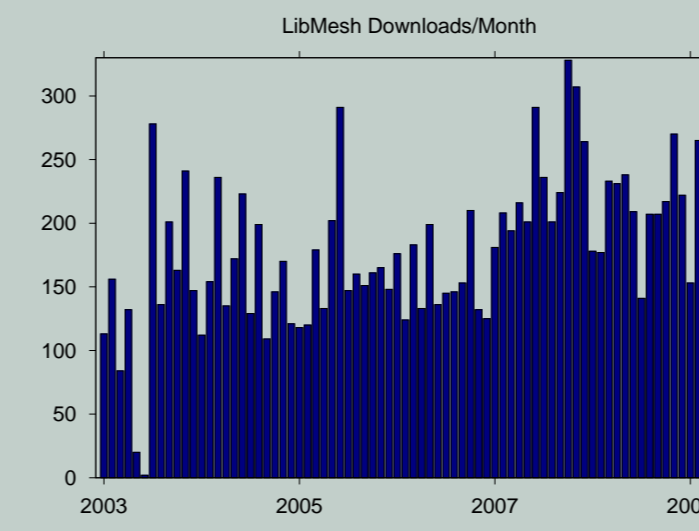
$$R(i) += k(xyz[q]) * (grad_u[q] * dphi[i][q]) * JxW[q];$$

References

- [KPSC06] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. *libMesh: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. Engineering With Computers*, 22(3):237–254, December 2006.

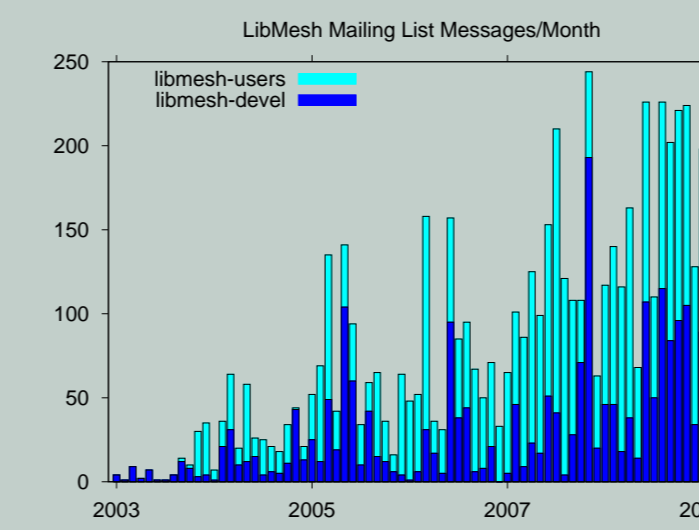
Thanks due to Benjamin Kirk, John Peterson, Vikram Garg, Varis Carey, Derek Gaston, Larisa Branets, Graham Carey, Paul Bauman, and many other contributors and collaborators.

Development



Open Source: libMesh is freely available from <http://libmesh.sf.net/>, and is redistributable under the GNU Lesser General Public License

Research: libMesh itself is the subject of one peer-reviewed paper [KPSC06], and has been used in the production of data for dozens of applications papers published by libMesh developers and users.



Open Development: design and development is made public in an international collaboration:

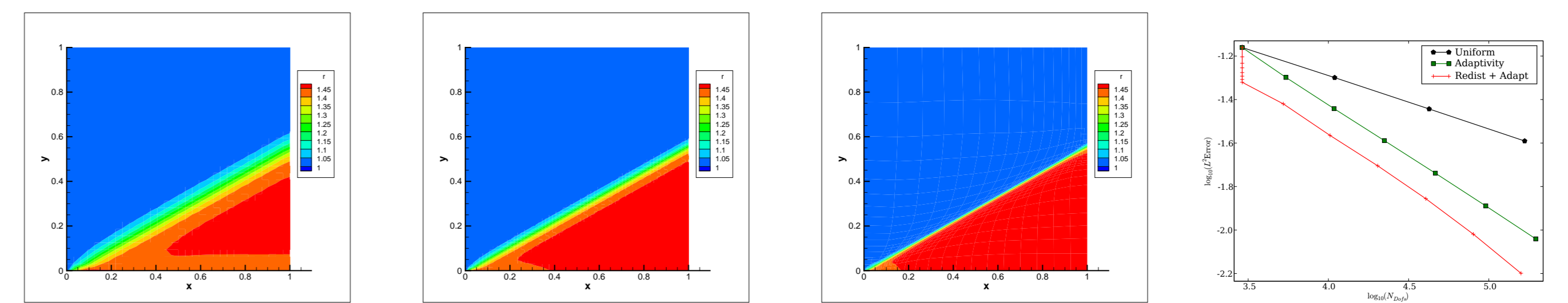
- Primary developers at UT, NASA, TACC, INL, MIT
- Recent code contributions from Germany, Switzerland, Brazil
- Public website, read access to Subversion repository
- Public `libmesh-users`, `libmesh-devel` mailing lists

Adaptivity

Error Estimation: *a posteriori* error estimators based on patch recovery, solution refinement, or jump residuals are available and applicable to a wide range of problems.

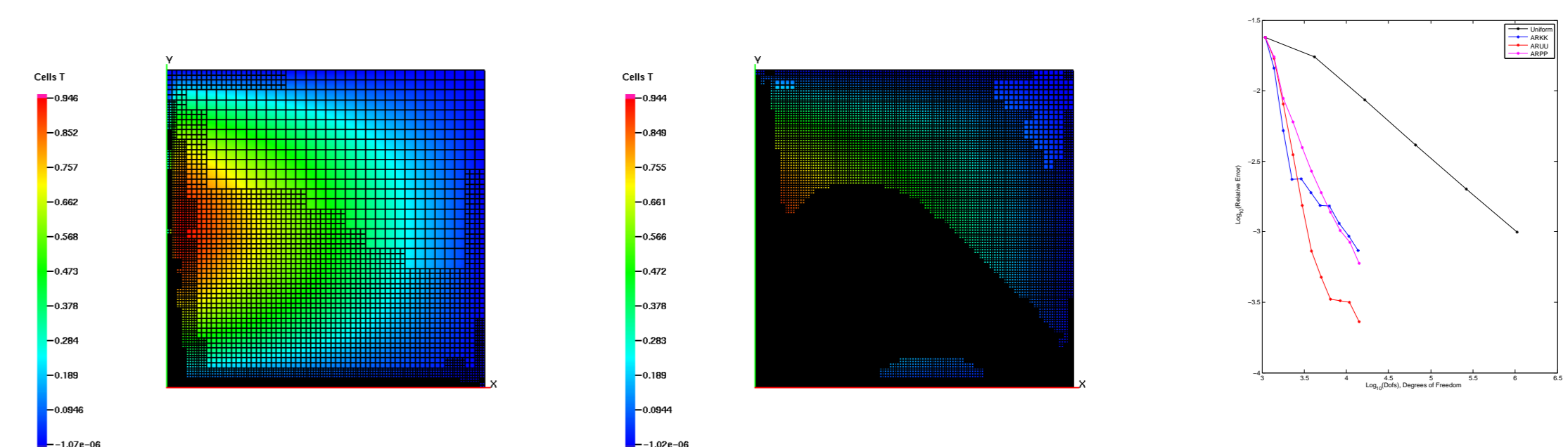
Adaptive Mesh Smoothing: libMesh integrates variational mesh optimization code, which can be applied using error estimation results to produce solution-adapted mesh redistribution.

Adaptive Mesh Refinement/Coarsening (AMR/C): natively supported, with non-conforming hierarchic meshes generated and hanging node continuity constraints automatically applied. Adaptive p refinement with hierarchic bases.



Adaptivity applied to a FIN-S perfect gas Euler benchmark problem, and global L_2 norm convergence

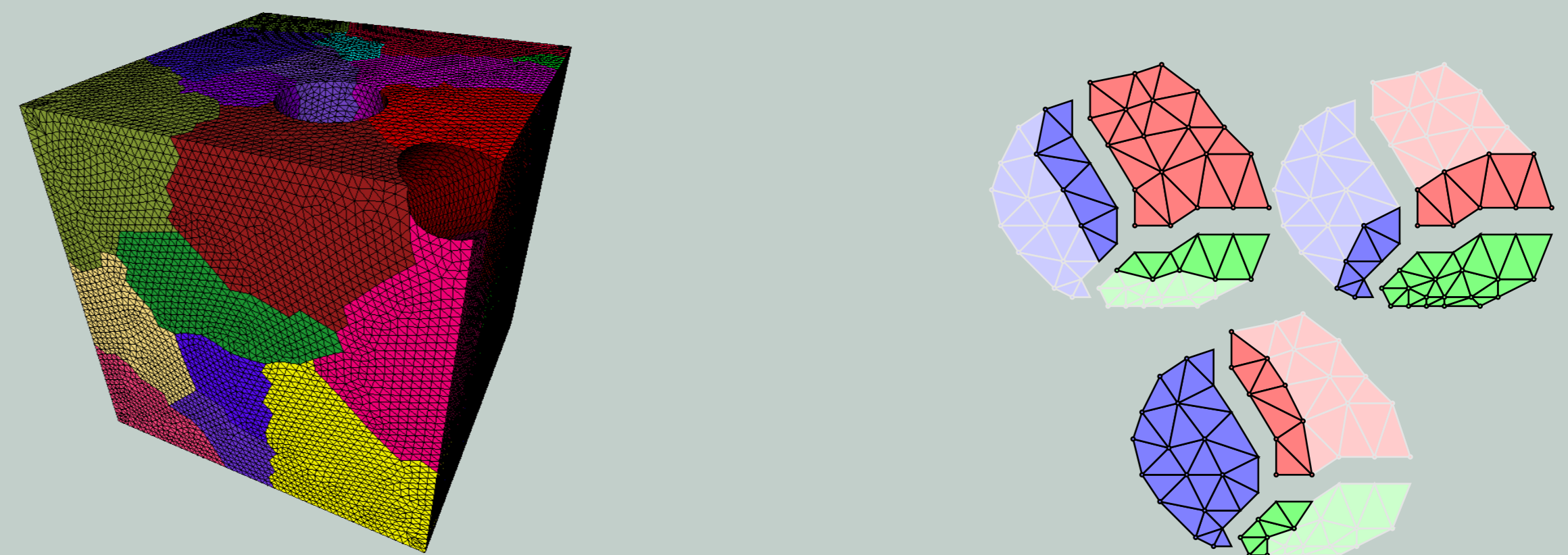
Goal-Oriented Adaptivity: global error norms are less important to PECOS than errors in local scalar quantity of interest (QoI) functionals. PECOS and collaborators are developing and verifying adjoint-based adaptivity capabilities, designed to specifically reduce error in a specified QoI.



Goal-oriented adaption with a solution layer and flux QoI on different boundary sides

Parallelism

Parallel Execution: Each distributed-memory Message Passing Interface (MPI) process performs calculations on its own element set. On hybrid parallel architectures, elements can be further subpartitioned for individual shared-memory threads using **Threading Building Blocks (TBB)**.



Domain (Re-)Partitioning: Meshes are automatically partitioned for the current number of processors, reconstructed when necessary for partitioning-independent I/O, and repartitioned for load balancing after AMR/C.

Data Synchronization: Ghost cells are automatically located. Neighboring processors communicate solution coefficients, constraint equations, etc. as required.

Adjoint-based Sensitivities

Forward Problem: (nonlinear)

$$\mathcal{R}(u_h(p), v_h; p) \equiv 0 \quad \forall v_h$$

Adjoint Problem: (linear!)

$$\frac{\partial \mathcal{R}(u_h, \phi(u_h, p); p)}{\partial u_h} \equiv \frac{\partial Q(u_h; p)}{\partial u_h}$$

Adjoint-based Sensitivity:

$$\frac{d\mathcal{R}}{dp} = \frac{\partial \mathcal{R}}{\partial p} + \frac{\partial \mathcal{R}}{\partial u_h} \frac{\partial u_h}{\partial p} = 0$$

$$\frac{dQ(u_h; p)}{dp} = \frac{\partial Q}{\partial p} + \frac{\partial Q}{\partial u_h} \frac{\partial u_h}{\partial p}$$

$$= \frac{\partial Q}{\partial p} - \frac{\partial \mathcal{R}(u_h, \phi)}{\partial p}$$

Application-Independent: Automatic adjoint calculations can be run on any `LinearImplicitSystem`, any `FEMSystem`, or any `NonlinearImplicitSystem`. Application code must assemble:

- QoI scalar $Q(u_h)$
- QoI derivative vector $\frac{\partial Q}{\partial u_h}$
- Jacobian matrix $\frac{\partial \mathcal{R}}{\partial u_h}$

Or use `FEMSystem` finite differenced Jacobians

Efficient: Requires one linear adjoint solve per QoI, one dot product per parameter. Fully finite differenced sensitivities require one nonlinear forward solve per parameter!