LibMesh

Roy H. Stogner      John W. Peterson

rostgnr@cfdlab.ae.utexas.edu      peterson@cfdlab.ae.utexas.edu

Univ. of Texas at Austin

June 6, 2007
Outline

In this talk we will discuss:

- Goals and example applications of the libMesh library.
- Some basic steps in writing a simple libMesh application.
- Design frameworks for more complex applications.
- Adaptive Mesh Refinement.
In this talk we will discuss:

- Goals and example applications of the libMesh library.
- Some basic steps in writing a simple libMesh application.
- Design frameworks for more complex applications.
- Adaptive Mesh Refinement.
In this talk we will discuss:

- Goals and example applications of the libMesh library.
- Some basic steps in writing a simple libMesh application.
- Design frameworks for more complex applications.
- Adaptive Mesh Refinement.
Outline

In this talk we will discuss:

- Goals and example applications of the libMesh library.
- Some basic steps in writing a simple libMesh application.
- Design frameworks for more complex applications.
- Adaptive Mesh Refinement.
libMesh is not:

- A physics implementation.
- A stand-alone application.

libMesh is:

- A software library and toolkit.
- Objects and functions for writing parallel adaptive finite element applications.
- An interface to linear algebra, meshing, partitioning, etc. libraries.
Goals

libMesh is not:

- A physics implementation.
- A stand-alone application.

libMesh is:

- A software library and toolkit.
- Objects and functions for writing parallel adaptive finite element applications.
- An interface to linear algebra, meshing, partitioning, etc. libraries.
Goals

libMesh is not:

- A physics implementation.
- A stand-alone application.

libMesh is:

- A software library and toolkit.
- Objects and functions for writing parallel adaptive finite element applications.
- An interface to linear algebra, meshing, partitioning, etc. libraries.
Goals

libMesh is not:
- A physics implementation.
- A stand-alone application.

libMesh is:
- A software library and toolkit.
- Objects and functions for writing parallel adaptive finite element applications.
- An interface to linear algebra, meshing, partitioning, etc. libraries.
Goals

libMesh is not:

- A physics implementation.
- A stand-alone application.

libMesh is:

- A software library and toolkit.
- Objects and functions for writing parallel adaptive finite element applications.
- An interface to linear algebra, meshing, partitioning, etc. libraries.
For this talk we will assume there is a mathematical model (Partial Differential Equation) to be solved in an engineering analysis:

\[
\begin{align*}
\frac{\partial u}{\partial t} &= \mathcal{R}(u) \in \Omega \\
u &= u_D \in \partial \Omega_D \\
\nabla u \cdot n &= u_N \in \partial \Omega_N
\end{align*}
\]
• Associated to the problem domain $\Omega$ is a libMesh data structure called a Mesh

• A Mesh is essentially a collection of finite elements

$$\Omega^h := \bigcup_e \Omega_e$$
Associated to the problem domain $\Omega$ is a libMesh data structure called a Mesh.

A Mesh is essentially a collection of finite elements

$$\Omega^h := \bigcup_e \Omega_e$$

libMesh provides some simple structured mesh generation routines as well as an interface to Triangle.
Compressible Gas Flow

- Mach 3 flow over a forward facing step.
Wave propagation from depth-averaged equations
Natural Convection

- Tetrahedral mesh of "pipe" geometry. Stream ribbons colored by temperature.
Surface-Tension-Driven Flow

Adaptive grid solution shown with temperature contours and velocity vectors.
Double-Diffusive Convection

- Solute contours: a plume of warm, low-salinity fluid is convected upward through a porous medium.
The tumor secretes a chemical which stimulates blood vessel formation.
Quenching separates fluid or alloy mixtures into multiple material phases.
The point of departure in any FE analysis which uses LibMesh is the weighted residual statement

\[(R(u), v) = 0 \quad \forall v \in V\]

Or, more precisely, the weighted residual statement associated with the finite-dimensional space \(V^h \subset V\)

\[(R(u^h), v^h) = 0 \quad \forall v^h \in V^h\]
The point of departure in any FE analysis which uses LibMesh is the weighted residual statement

\[(\mathcal{R}(u), v) = 0 \quad \forall v \in V\]

Or, more precisely, the weighted residual statement associated with the finite-dimensional space \(V^h \subset V\)

\[(\mathcal{R}(u^h), v^h) = 0 \quad \forall v^h \in V^h\]
Some Examples

Poisson Equation

$$-\Delta u = f \quad \in \Omega$$
Some Examples

Poisson Equation

\[-\Delta u = f \quad \in \quad \Omega\]

Weighted Residual Statement

\[
(\mathcal{R}(u), v) := \int_{\Omega} [\nabla u \cdot \nabla v - fv] \, dx \\
+ \int_{\partial \Omega_N} (\nabla u \cdot \mathbf{n}) v \, ds
\]
Some Examples

Linear Convection-Diffusion

\[-k\Delta u + b \cdot \nabla u = f \quad \in \Omega\]
Some Examples

Linear Convection-Diffusion

\[-k \Delta u + b \cdot \nabla u = f \quad \in \quad \Omega\]

Weighted Residual Statement

\[ (\mathcal{R}(u), v) := \int_{\Omega} [k \nabla u \cdot \nabla v + (b \cdot \nabla u)v -fv] \, dx \]

\[ + \int_{\partial\Omega_N} k (\nabla u \cdot n) v \, ds \]
Some Examples

Stokes Flow

\[
\nabla p - \nu \Delta u = f \\
\n\nabla \cdot u = 0
\n\in \Omega
\]

\[\n\]
Some Examples

Stokes Flow

\[ \nabla p - \nu \Delta u = f \]
\[ \nabla \cdot u = 0 \quad \in \Omega \]

Weighted Residual Statement

\[ u := [u, p] \quad , \quad v := [v, q] \]

\[ (\mathcal{R}(u), v) := \int_{\Omega} \left[ -p (\nabla \cdot v) + \nu \nabla u : \nabla v - f \cdot v \right. \]
\[ \left. + (\nabla \cdot u) q \right] dx + \int_{\partial \Omega_N} \left( \nu \nabla u - pI \right) \cdot n \cdot v \, ds \]
To obtain the approximate problem, we simply replace $u \leftarrow u^h$, $v \leftarrow v^h$, and $\Omega \leftarrow \Omega^h$ in the weighted residual statement.
For simplicity we will focus on the weighted residual statement arising from the Poisson equation, with \( \partial \Omega_N = \emptyset \),

\[
\left( \mathcal{R}(u^h), v^h \right) := \int_{\Omega^h} \left[ \nabla u^h \cdot \nabla v^h - f v^h \right] \, dx = 0 \quad \forall v^h \in V^h
\]
The integral over $\Omega^h$ ...

$$0 = \int_{\Omega^h} \left[ \nabla u^h \cdot \nabla v^h - f v^h \right] \, dx \quad \forall v^h \in V^h$$
The integral over $\Omega^h$ . . . is written as a sum of integrals over the $N_e$ finite elements:

$$0 = \int_{\Omega^h} \left[ \nabla u^h \cdot \nabla v^h - f v^h \right] \, dx \quad \forall v^h \in \mathcal{V}^h$$

$$= \sum_{e=1}^{N_e} \int_{\Omega_e} \left[ \nabla u^h \cdot \nabla v^h - f v^h \right] \, dx \quad \forall v^h \in \mathcal{V}^h$$
An element integral will have contributions only from the global basis functions corresponding to its nodes.

We call these local basis functions $\phi_i$, $0 \leq i \leq N_s$.

$$v^h \big|_{\Omega_e} = \sum_{i=1}^{N_s} c_i \phi_i$$
An element integral will have contributions only from the global basis functions corresponding to its nodes. We call these local basis functions $\phi_i$, $0 \leq i \leq N_s$.

\[ v^h \big|_{\Omega_e} = \sum_{i=1}^{N_s} c_i \phi_i \]

\[ \int_{\Omega_e} v^h \, dx = \sum_{i=1}^{N_s} c_i \int_{\Omega_e} \phi_i \, dx \]
The element integrals . . .

\[ \int_{\Omega_e} \left[ \nabla u^h \cdot \nabla v^h - f v^h \right] dx \]
The element integrals . . .

\[ \int_{\Omega_e} \left[ \nabla u^h \cdot \nabla v^h - f v^h \right] dx \]

are written in terms of the local “\( \phi_i \)” basis functions

\[ \sum_{j=1}^{N_s} u_j \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \; dx - \int_{\Omega_e} f \phi_i \; dx \; , \; \; i = 1, \ldots, N_s \]
The element integrals ... 

\[ \int_{\Omega_e} \left( \nabla u^h \cdot \nabla v^h - f v^h \right) \, dx \]

are written in terms of the local "\( \phi_i \)" basis functions

\[ \sum_{j=1}^{N_s} u_j \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx - \int_{\Omega_e} f \phi_i \, dx , \quad i = 1, \ldots, N_s \]

This can be expressed naturally in matrix notation as

\[ K^e U^e - F^e \]
The entries of the element stiffness matrix are the integrals

\[ K_{ij}^e := \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx \]
The entries of the element stiffness matrix are the integrals

\[ K^e_{ij} := \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx \]

While for the element right-hand side we have

\[ F^e_i := \int_{\Omega_e} f \phi_i \, dx \]
The entries of the element stiffness matrix are the integrals

\[ K_{ij}^e := \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx \]

While for the element right-hand side we have

\[ F_i^e := \int_{\Omega_e} f \phi_i \, dx \]

The element stiffness matrices and right-hand sides can be “assembled” to obtain the global system of equations

\[ KU = F \]
The integrals are performed on a “reference” element $\hat{\Omega}_e$.
The integrals are performed on a “reference” element $\hat{\Omega}_e$.

The Jacobian of the map $x(\xi)$ is $J$.

\[ F_i^e = \int_{\Omega_e} f \phi_i dx = \int_{\hat{\Omega}_e} f(x(\xi)) \phi_i |J| d\xi \]
The integrals are performed on a “reference” element \( \hat{\Omega}_e \)

Chain rule: \( \nabla = J^{-1} \nabla_\xi := \hat{\nabla}_\xi \).

\[
K_{ij}^e = \int_{\Omega_e} \nabla \phi_j \cdot \nabla \phi_i \, dx = \int_{\hat{\Omega}_e} \hat{\nabla}_\xi \phi_j \cdot \hat{\nabla}_\xi \phi_i \ |J| \, d\xi
\]
The integrals on the “reference” element are approximated via numerical quadrature.
The integrals on the “reference” element are approximated via numerical quadrature.

The quadrature rule has $N_q$ points “$\xi_q$” and weights “$w_q$”. 
The integrals on the “reference” element are approximated via numerical quadrature.

The quadrature rule has $N_q$ points “$\xi_q$” and weights “$w_q$”.

\[
F_i^e = \int_{\hat{\Omega}_e} f \phi_i |J| d\xi 
\approx \sum_{q=1}^{N_q} f(x(\xi_q)) \phi_i(\xi_q) |J(\xi_q)| w_q
\]
The integrals on the “reference” element are approximated via numerical quadrature.

The quadrature rule has $N_q$ points “$\xi_q$” and weights “$w_q$”.

$$K_{ij}^e = \int_{\hat{\Omega}_e} \hat{\nabla}_{\xi} \phi_j \cdot \hat{\nabla}_{\xi} \phi_i |J| d\xi$$

$$\approx \sum_{q=1}^{N_q} \hat{\nabla}_{\xi} \phi_j(\xi_q) \cdot \hat{\nabla}_{\xi} \phi_i(\xi_q) |J(\xi_q)| w_q$$
LibMesh provides the following variables at each quadrature point \( q \)

<table>
<thead>
<tr>
<th>Code</th>
<th>Math</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JxW[(q)]</td>
<td>(</td>
<td>J(\xi_q)</td>
</tr>
<tr>
<td>phi[i][(q)]</td>
<td>( \phi_i(\xi_q) )</td>
<td>value of ( i^{th} ) shape fn.</td>
</tr>
<tr>
<td>dphi[i][(q)]</td>
<td>( \nabla_{\xi} \phi_i(\xi_q) )</td>
<td>value of ( i^{th} ) shape fn. gradient</td>
</tr>
<tr>
<td>xyz[(q)]</td>
<td>( x(\xi_q) )</td>
<td>location of ( \xi_q ) in physical space</td>
</tr>
</tbody>
</table>
The **LibMesh** representation of the matrix and rhs assembly is similar to the mathematical statements.

```c
for (q=0; q<Nq; ++q)
    for (i=0; i<Ns; ++i) {
        Fe(i) += JxW[q] * f(xyz[q]) * phi[i][q];

    for (j=0; j<Ns; ++j)
        Ke(i,j) += JxW[q] * (dphi[j][q] * dphi[i][q]);
    }
```
The LibMesh representation of the matrix and rhs assembly is similar to the mathematical statements.

\[
\text{for } (q=0; \ q<Nq; \ ++q) \\
\quad \text{for } (i=0; \ i<Ns; \ ++i) \ { \\
\quad \quad \text{Fe}(i) \ += \ JxW[q] \times f(xyz[q]) \times \phi[i][q]; \\
\quad }
\]

\[
\text{for } (j=0; \ j<Ns; \ ++j) \\
\quad \text{Ke}(i, j) \ += \ JxW[q] \times (dphi[j][q] \times dphi[i][q]);
\]

\[
F^e_i = \sum_{q=1}^{N_q} f(x(\xi_q)) \phi_i(\xi_q) |J(\xi_q)| w_q
\]
The LibMesh representation of the matrix and rhs assembly is similar to the mathematical statements.

\[
\begin{align*}
\text{for (q=0; q<Nq; ++q) } \\
\quad \text{for (i=0; i<Ns; ++i) } \\
\quad \quad \text{Fe}(i) &= \text{JxW}[q] \times f(xyz[q]) \times \text{phi}[i][q]; \\
\quad \text{for (j=0; j<Ns; ++j) } \\
\quad \quad \text{Ke}(i,j) &= \text{JxW}[q] \times (\text{dphi}[j][q] \times \text{dphi}[i][q]); \\
\end{align*}
\]

\[
F_i^e = \sum_{q=1}^{N_q} f(x(\xi_q)) \phi_i(\xi_q) |J(\xi_q)|w_q
\]
The **LibMesh** representation of the matrix and rhs assembly is similar to the mathematical statements.

```c
for (q=0; q<Nq; ++q)
    for (i=0; i<Ns; ++i) {
        Fe(i) += JxW[q] * f(xyz[q]) * phi[i][q];
    }

for (j=0; j<Ns; ++j)
    Ke(i, j) += JxW[q] * (dphi[j][q] * dphi[i][q]);
```

\[
F^e_i = \sum_{q=1}^{N_q} f(x(\xi_q)) \phi_i(\xi_q) |J(\xi_q)| w_q
\]
The LibMesh representation of the matrix and rhs assembly is similar to the mathematical statements.

\[
\text{for (q=0; q<Nq; ++q)} \\
\quad \text{for (i=0; i<Ns; ++i) } \{ \\
\quad \quad \text{Fe}(i) += JxW[q] \times f(xyz[q]) \times \phi[i][q]; \\
\quad \}\n\]

\[
\text{for (j=0; j<Ns; ++j)} \\
\quad \text{Ke}(i, j) += JxW[q] \times (d\phi[j][q] \times d\phi[i][q]); \\
\}
\]

\[
F_i^e = \sum_{q=1}^{N_q} f(x(\xi_q)) \phi_i(\xi_q) |J(\xi_q)| w_q
\]
The **LibMesh** representation of the matrix and rhs assembly is similar to the mathematical statements.

```c
for (q=0; q<Nq; ++q)
    for (i=0; i<Ns; ++i) {
        Fe(i) += JxW[q]*f(xyz[q])*phi[i][q];
    }

for (j=0; j<Ns; ++j)
    Ke(i, j) += JxW[q]*(dphi[j][q]*dphi[i][q]);
```

\[
K_{ij}^e = \sum_{q=1}^{N_q} \hat{\nabla}_\xi \phi_j(\xi_q) \cdot \hat{\nabla}_\xi \phi_i(\xi_q) |J(\xi_q)| w_q
\]
The \textbf{LibMesh} representation of the matrix and rhs assembly is similar to the mathematical statements.

\begin{equation}
\begin{aligned}
\text{for } (q=0; \ q<Nq; \ ++q) \\
\quad \text{for (i=0; \ i<Ns; \ ++i) } \\
\quad \quad \text{Fe}(i) \quad += \ JxW[q] \times f(xyz[q]) \times \phi[i][q]; \\
\end{aligned}
\end{equation}

\begin{equation}
\begin{aligned}
\text{for (j=0; \ j<Ns; \ ++j) } \\
\quad \text{Ke}(i, j) \quad += \ JxW[q] \times (d\phi[j][q] \times d\phi[i][q]); \\
\end{aligned}
\end{equation}

\begin{equation}
K_{ij}^e = \sum_{q=1}^{N_q} \nabla_\xi \phi_j(\xi_q) \cdot \nabla_\xi \phi_i(\xi_q) |J(\xi_q)| w_q
\end{equation}
The LibMesh representation of the matrix and rhs assembly is similar to the mathematical statements.

\[
\text{for } (q=0; \ q<Nq; \ ++q) \\
\quad \text{for } (i=0; \ i<Ns; \ ++i) \ { \\
\qquad \text{Fe}(i) \ += \ JxW[q]*f(xyz[q])*phi[i][q]; \\
\quad \} \\
\quad \text{for } (j=0; \ j<Ns; \ ++j) \\
\qquad \text{Ke}(i,j) \ += \ JxW[q]*(dphi[j][q]*dphi[i][q]); \\
\}
\]

\[
K_{ij}^e = \sum_{q=1}^{Nq} \hat{\nabla}_\xi \phi_j(\xi_q) \cdot \hat{\nabla}_\xi \phi_i(\xi_q) |J(\xi_q)| w_q
\]
Object Oriented Programming

- Abstract Base Classes define user interfaces.
- Concrete Subclasses implement functionality.
- One physics code can work with many discretizations.
Object Oriented Programming

- Abstract Base Classes define user interfaces.
- Concrete Subclasses implement functionality.
- One physics code can work with many discretizations.
Object Oriented Programming

- Abstract Base Classes define user interfaces.
- Concrete Subclasses implement functionality.
- One physics code can work with many discretizations.
Object Oriented Programming

- Abstract Base Classes define user interfaces.
- Concrete Subclasses implement functionality.
- One physics code can work with many discretizations.
Abstract interface gives mesh topology

Concrete instantiations of mesh geometry

Hides element type from most applications
Finite Element Classes

- Finite Element object builds data for each Geometric object
- User only deals with shape function, quadrature data
For linear problems, we have already seen how the weighted residual statement leads directly to a sparse linear system of equations

\[ KU = F \]
For time-dependent problems,

\[ \frac{\partial u}{\partial t} = \mathcal{R}(u) \]

we also need a way to advance the solution in time, e.g. a \( \theta \)-method

\[
\left( \frac{u^{n+1} - u^n}{\Delta t}, v^h \right) = \left( \mathcal{R}(u_\theta), v^h \right) \quad \forall v^h \in \mathcal{V}^h
\]

\[
u_\theta := \theta u^{n+1} + (1 - \theta)u^n
\]

Leads to \( KU = F \) at each timestep.
For nonlinear problems, typically a sequence of linear problems must be solved, e.g. for Newton’s method

\[(\mathcal{R}'(u^h)\delta u^h, v^h) = -(\mathcal{R}(u^h), v^h)\]

where \(\mathcal{R}'(u^h)\) is the linearized (Jacobian) operator associated with the PDE.

Must solve \(KU = F\) (Inexact Newton method) at each iteration step.
Boundary Value Problem Framework Goals

Goals:

- Improving test coverage and reliability
- Hiding of implementation details from user code
- Rapid prototyping of differential equation approximations
- Improved error estimation

Methods:

- Object-oriented System and Solver classes
- Numerical Jacobian verification
**FEM System Classes**

- Generalized IBVP representation
- FEMSystem does all initialization, global assembly
- User code only needs weighted time derivative and/or constraint functions
**ODE Solver Classes**

- **TimeSolver**
  - `_residual(request_jacobian)`
  - `solve()`
  - `advance_timestep()`

  - **SteadySolver**
  - **EulerSolver**

  - **AdamsMoultonSolver**
  - **EigenSolver**

- Calls user code on each element
- Assembles element-by-element time derivatives, constraints, and weighted old solutions
Nonlinear Solver Classes

- Acquires residuals, jacobians from ODE solver
- Handles inner loops, inner solvers and tolerances, convergence tests, etc
1D refinement example

- Consider the 1D model convection-diffusion equation

\[
\begin{align*}
-u'' + bu' & = 0 \quad \forall \ 0 \leq x \leq 1 \\
u(0) & = 0 \\
u(1) & = 1
\end{align*}
\]

- The convection-diffusion equation can be thought of as a particularly simple form of the drift-diffusion equation.
- The exact solution is

\[
u = \frac{1 - \exp(bx)}{1 - \exp(b)}\]
For large values of $b$, the solution changes rapidly near $x = 1$.

The solution for $b = 10$. 
We assume here that we have an approximate solution \( u_h \) which is the \textit{linear interpolant} of \( u \).

We will measure the error between the exact solution \( u \) and the approximate solution \( u_h \) in the following \((L_2)\) norm:

\[
\| e \|_{L_2}^2 := \| u - u_h \|_{L_2}^2 = \int_0^1 (u - u_h)^2 \, dx
\]

Consider a sequence of uniformly-refined meshes . . .
4 elements, $\|e\|_{L_2} = 0.09$
8 elements, $\|e\|_{L^2} = 0.027$
16 elements, $\|e\|_{L^2} = 0.0071$
Adaptive Refinement

Q: Can we do “better” than uniform refinement?
A: Yes, if we refine cells which have higher error relative to others.
In general we don’t know the exact solution, and so we need a way of *estimating* the error.

Consider the following formula for estimating the error, $\eta$, in an element defined on the interval $(x_i, x_{i+1})$

$$\eta^2 := \frac{h}{2} \sum_{k=i}^{i+1} \left[ u'(x_k) \right]^2$$

where $h := x_{i+1} - x_i$ is the element length, and

The “jump” in $u'$ is

$$\left[ u'(x_k) \right] := \left| u'(x_k^+) - u'(x_k^-) \right|$$
Error Indicator, Uniformly-Refined Grids

4 elements
Error Indicator, Uniformly-Refined Grids

8 elements
Error Indicator, Uniformly-Refined Grids

16 elements
A simple adaptive refinement strategy with \( r_{\text{max}} \) refinement steps for this 1D example problem is:

\[
\begin{align*}
  r &= 0; \\
  \text{while } (r < r_{\text{max}}) \\
  &\quad \text{Compute the FE solution (linear interpolant)} \\
  &\quad \text{Estimate the error (using flux-jump indicator)} \\
  &\quad \text{Refine the element with highest error} \\
  &\quad \text{Increment } r \\
  \end{align*}
\]
2 elements
8 elements
The graph shows the comparison between two functions, $u$ and $u_h$, over a domain from $x=0$ to $x=1$. The graph is labeled with '9 elements' indicating the number of elements used in the finite element method. The blue line represents $u$, and the red line represents $u_h$.
Introduction

Applications

Weighted Residuals

Finite Elements

Essential BCs

Complex Problems

Adaptivity

Graph showing a comparison between $u$ and $u_h$ with 10 elements.
Final: 13 elements
Error Plot vs. Number of Nodes

- Logarithmic scale for both axes: $\log_{10} N_{\text{nodes}}$ vs. $\log_{10} ||e||$
- Two lines: Uniform and Adaptive
- The Adaptive line shows a steeper decrease in error with an increase in the number of nodes compared to the Uniform line.
Non-conforming meshes lead to “hanging nodes”, and to provide continuous finite element function the fine element degrees of freedom must be constrained in terms of coarse element degrees of freedom.

\[ u^F = u^C \]

\[ \sum_i u_i^F \phi_i^F = \sum_j u_j^C \phi_j^C \]

\[ u_i^F = C_{ij} u_j^C \]
Adaptive $p$ Constraints

- $p$ refinement is done with hierarchic adaptivity
- Hanging degree of freedom coefficients are simply set to 0
Diffuse Interface Modeling with AMR/C

- Mesh coarsening in smooth regions is traded for mesh refinement in sharp layers
- Equivalent accuracy is achieved here with 75% fewer degrees of freedom than a uniform mesh
Installing libMesh

PETSc Download, Installation instructions, Documentation:

libMesh Downloads, Installation instructions, Documentation:
http://libmesh.sourceforge.net/