Refactoring Finite Element Computation

Matthew Knepley

Mathematics and Computer Science Division
Argonne National Laboratory

CI Talk
University of Chicago
March 20, 2008

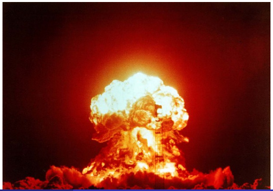
Necessity Of Simulation
Experiment are ...

Expensive

Difficult

Impossible

Dangerous



What is the Way Forward?

How do we move Scientific Computing forward?

- Performance
  - Bandwidth on multicore chips
- Experimentation
  - Solvers (solved)
  - Elements
  - Models
- Coupling
  - How does this interact with the discretization. . .
  - or solver?

Outline

Problems

The biggest problem in scientific computing is programmability:

- Lack of usable implementations of modern algorithms
  - Unstructured Multigrid
  - Fast Multipole Method
- Lack of comparison among classes of algorithms
  - Meshes
  - Discretizations

We should reorient thinking from

- characterizing the solution (FEM)
  - “what is the convergence rate (in h) of this finite element?”

to

- characterizing the computation (FErari)
  - “how many digits of accuracy per flop for this finite element?”

Problems

The biggest problem in scientific computing is programmability:

- Lack of widespread implementations of modern algorithms
    - Unstructured Multigrid
    - Fast Multipole Method
- Lack of comparison among classes of algorithms
    - Meshes
    - Discretizations

We should reorient thinking from

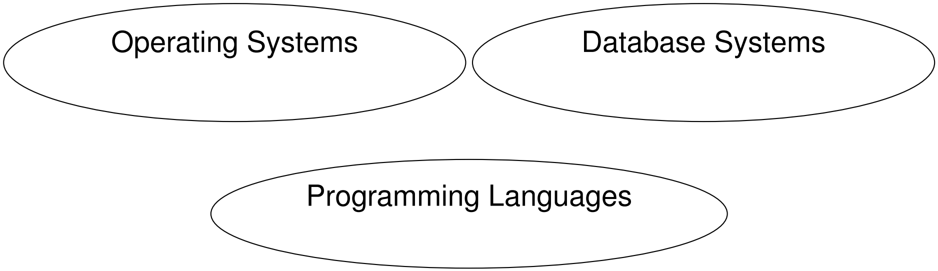
- characterizing the solution (FEM)
    - “what is the convergence rate (in *h*) of this finite element?”

to

- characterizing the computation (FErari)
    - “how many digits of accuracy per flop for this finite element?”

Interaction with Systems

We have to bridge the gap with Systems
to enable Scientific Computing



Interaction with Systems

We have to bridge the gap with Systems
to enable Scientific Computing

Operating Systems
Distributed Computing

Database Systems

Programming Languages

Interaction with Systems

We have to bridge the gap with Systems
to enable Scientific Computing

Operating Systems
Distributed Computing

Database Systems
Datamining

Programming Languages

Interaction with Systems

We have to bridge the gap with Systems
to enable Scientific Computing

Operating Systems
Distributed Computing

Database Systems
Datamining

Programming Languages
Code Generation

Future Compilers

I think compilers are victims of their own success (ala Rob Pike)

- Efforts to modularize compilers retain the same primtives
    - compiling on the fly (JIT)
    - **L**ow **L**evel **V**irtual **M**achine
- Raise the level of abstraction
    - **F**enics **F**orm **C**ompiler (variational form compiler)
    - Mython (**D**omain **S**pecific **L**anguage generator)

Representation Hierarchy

Divide the work into levels:

- Model
- Algorithm
- Implementation

Representation Hierarchy

Divide the work into levels:

- Model
- Algorithm
- Implementation

Spiral Project:

- **D**iscrete **F**ourier **T**ransform (DSP)
- **F**ast **F**ourier **T**ransform (SPL)
- C Implementation (SPL Compiler)

Representation Hierarchy

Divide the work into levels:

- Model
- Algorithm
- Implementation

FLAME Project:

- Abstract LA (PME/Invariants)
- Basic LA (FLAME/FLASH)
- Scheduling (SuperMatrix)

Representation Hierarchy

Divide the work into levels:

- Model
- Algorithm
- Implementation

FEniCS Project:

- Navier-Stokes (FFC)
- Finite Element (FIAT)
- Integration/Assembly (FErari)

Representation Hierarchy

Divide the work into levels:

- Model
- Algorithm
- Implementation

Treecodes:

- Kernels with decay (Coulomb)
- Treecodes (PetFMM)
- Scheduling (PetFMM-GPU)

Representation Hierarchy

Divide the work into levels:

- Model
- Algorithm
- Implementation

Treecodes:

- Kernels with decay (Coulomb)
- Treecodes (PetFMM)
- Scheduling (PetFMM-GPU)

Each level demands a strong abstraction layer

Outline

Outline

Form Decomposition

Element integrals are decomposed into analytic and geometric parts:

$$
\begin{aligned}
& \int_{\mathcal{T}} \nabla\phi_i(\mathbf{x}) \cdot \nabla\phi_j(\mathbf{x}) d\mathbf{x} && (1)\\
= & \int_{\mathcal{T}} \frac{\partial\phi_i(\mathbf{x})}{\partial x_\alpha} \frac{\partial\phi_j(\mathbf{x})}{\partial x_\alpha} d\mathbf{x} && (2)\\
= & \int_{\mathcal{T}_{\mathrm{ref}}} \frac{\partial\xi_\beta}{\partial x_\alpha} \frac{\partial\phi_i(\xi)}{\partial\xi_\beta} \frac{\partial\xi_\gamma}{\partial x_\alpha} \frac{\partial\phi_j(\xi)}{\partial\xi_\gamma} |J| d\mathbf{x} && (3)\\
= & \frac{\partial\xi_\beta}{\partial x_\alpha} \frac{\partial\xi_\gamma}{\partial x_\alpha} |J| \int_{\mathcal{T}_{\mathrm{ref}}} \frac{\partial\phi_i(\xi)}{\partial\xi_\beta} \frac{\partial\phi_j(\xi)}{\partial\xi_\gamma} d\mathbf{x} && (4)\\
= & G^{\beta\gamma}(\mathcal{T}) K^{ij}_{\beta\gamma} && (5)
\end{aligned}
$$

Coefficients are also put into the geometric part.

Element Matrix Formation

- Element matrix K is now made up of small tensors
- Contract all tensor elements with each the geometry tensor $G(\mathcal{T})$

$\begin{matrix}3&0\\0&0\end{matrix}$	$\begin{matrix}0&-1\\0&0\end{matrix}$	$\begin{matrix}1&1\\0&0\end{matrix}$	$\begin{matrix}-4&-4\\0&0\end{matrix}$	$\begin{matrix}0&4\\0&0\end{matrix}$	$\begin{matrix}0&0\\0&0\end{matrix}$
$\begin{matrix}0&0\\-1&0\end{matrix}$	$\begin{matrix}0&0\\0&3\end{matrix}$	$\begin{matrix}0&0\\1&1\end{matrix}$	$\begin{matrix}0&0\\0&0\end{matrix}$	$\begin{matrix}0&0\\4&0\end{matrix}$	$\begin{matrix}0&0\\-4&-4\end{matrix}$
$\begin{matrix}1&0\\1&0\end{matrix}$	$\begin{matrix}0&1\\0&1\end{matrix}$	$\begin{matrix}3&3\\3&3\end{matrix}$	$\begin{matrix}-4&0\\-4&0\end{matrix}$	$\begin{matrix}0&0\\0&0\end{matrix}$	$\begin{matrix}0&-4\\0&-4\end{matrix}$
$\begin{matrix}-4&0\\-4&0\end{matrix}$	$\begin{matrix}0&0\\0&0\end{matrix}$	$\begin{matrix}-4&-4\\0&0\end{matrix}$	$\begin{matrix}8&4\\4&8\end{matrix}$	$\begin{matrix}0&-4\\-4&-8\end{matrix}$	$\begin{matrix}0&4\\4&0\end{matrix}$
$\begin{matrix}0&0\\4&0\end{matrix}$	$\begin{matrix}0&4\\0&0\end{matrix}$	$\begin{matrix}0&0\\0&0\end{matrix}$	$\begin{matrix}0&-4\\-4&-8\end{matrix}$	$\begin{matrix}8&4\\4&8\end{matrix}$	$\begin{matrix}-8&-4\\-4&0\end{matrix}$
$\begin{matrix}0&0\\0&0\end{matrix}$	$\begin{matrix}0&-4\\0&-4\end{matrix}$	$\begin{matrix}0&0\\-4&-4\end{matrix}$	$\begin{matrix}0&4\\4&0\end{matrix}$	$\begin{matrix}-8&-4\\-4&0\end{matrix}$	$\begin{matrix}8&4\\4&8\end{matrix}$

Element Matrix Computation

- Element matrix K can be precomputed
    - FFC
    - SyFi
- Can be extended to nonlinearities and curved geometry
- Many redundancies among tensor elements of K
    - Could try to optimize the tensor contraction...

Abstract Problem

Given vectors $v_i \in \mathbb{R}^m$, minimize *flops*$(v^T g)$ for arbitrary $g \in \mathbb{R}^m$

- The set v_i is not at all random
- Not a traditional compiler optimization
- How to formulate as an optimization problem?

Outline

Complexity Reducing Relations

If $v_i^T g$ is known, is $flops(v_j^T g) < 2m - 1$?

We can use binary relations among the vectors:

- Equality
  - If $v_j = v_i$, then $flops(v_j^T g) = 0$
- Colinearity
  - If $v_j = \alpha v_i$, then $flops(v_j^T g) = 1$
- Hamming distance
  - If $dist_H(v_j, v_i) = k$, then $flops(v_j^T g) = 2k$

Algorithm for Binary Relations

- Construct a weighted graph on v_i
  - The weight $w(i, j)$ is *flops*$(v_j^T g)$ given $v_i^T g$
  - With the above relations, the graph is symmetric
- Find a minimum spanning tree
  - Use Prim or Kruskal for worst case $O(n^2 \log n)$
- Traverse the MST, using the appropriate calculation for each edge
  - Roots require a full dot product

Coplanarity

- Ternary relation
  - If $v_k = \alpha v_i + \beta v_j$, then $flops(v_k^T g) = 3$
  - Does not fit our undirected graph paradigm
- SVD for vector triples is expensive
  - Use a randomized projection into a few $\mathbb{R}^3$s
- Use a hypergraph?
  - MST algorithm available
- Appeal to geometry?
  - Incidence structures

Outline

FErari

Finite Element rearragement to automaically reduce instructions

- Open source implementation http://www.fenics.org/wiki/FErari
- Build tensor blocks $K^{ij}_{m,m'}$ as vectors using FIAT
- Discover dependencies
  - Represented as a DAG
  - Can also use hypergraph model
- Use minimal spanning tree to construct computation

Preliminary Results

Order	Entries	Base MAPs	FErari MAPs
1	6	24	7
2	21	84	15
3	55	220	45
4	120	480	176
5	231	924	443
6	406	1624	867

Outline

Modeling the Problem

- Objective is cost of dot products (tensor contractions in FEM)
    - Set of vectors V with a given arbitrary vector g
- The original MINLP has a nonconvex, nonlinear objective
- Reformulate to obtain a MILP using auxiliary binary variables

Modeling the Problem

Variables

$$\begin{aligned}\alpha_{ij} &= \text{Basis expansion coefficients}\\ y_i &= \text{Binary variable indicating membership in the basis}\\ s_{ij} &= \text{Binary variable indicating nonzero coefficient } \alpha_{ij}\\ z_{ij} &= \text{Binary variable linearizes the objective function (equivalent to } y_i y_j\text{)}\\ U &= \text{Upper bound on coefficients}\end{aligned}$$

Constraints

$$\begin{aligned}Eq.\ (6b) &: \text{Basis expansion}\\ Eq.\ (6c) &: \text{Exclude nonbasis vector from the expansion}\\ Eq.\ (6d) &: \text{Remove offdiagonal coefficients for basis vectors}\\ Eq.\ (7c) &: \text{Exclude vanishing coefficients from cost}\end{aligned}$$

Original Formulation

MINLP Model

$$\text{minimize} \quad \sum_{i=1}^{n} \left\{ y_i(2m-1) + (1-y_i)\left(2\sum_{j=1,j\neq i}^{n} y_j - 1\right)\right\} \tag{6a}$$

$$\text{subject to} \quad v_i = \sum_{j=1}^{n} \alpha_{ij} v_j \qquad i = 1,\ldots,n \tag{6b}$$

$$-Uy_j \leq \alpha_{ij} \leq Uy_j \qquad i,j = 1,\ldots,n \tag{6c}$$

$$-U(1-y_i) \leq \alpha_{ij} \leq U(1-y_i) \qquad i,j = 1,\ldots,n, \tag{6d}$$

$$y_i \in \{0,1\} \qquad i = 1,\ldots,n. \tag{6e}$$

Original Formulation

Equivalent MILP Model: $z_{ij} = y_i \cdot y_j$

$$\text{minimize} \quad 2m\sum_{i=1}^{n} y_i + 2\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}(y_j - z_{ij}) - n \tag{6a}$$

$$\text{subject to} \quad v_i = \sum_{j=1}^{n}\alpha_{ij}v_j \qquad i = 1,\ldots,n \tag{6b}$$

$$-Uy_j \leq \alpha_{ij} \leq Uy_j \qquad i,j = 1,\ldots,n \tag{6c}$$

$$-U(1-y_i) \leq \alpha_{ij} \leq U(1-y_i) \qquad i,j = 1,\ldots,n,\ i\neq j \tag{6d}$$

$$z_{ij} \leq y_i, \quad z_{ij} \leq y_j, \quad z_{ij} \geq y_i + y_j - 1, \qquad i,j = 1,\ldots,n \tag{6e}$$

$$y_i \in \{0,1\}, \quad z_{ij} \in \{0,1\} \qquad i,j = 1,\ldots,n.$$

Sparse Coefficient Formulation

- Take advantage of sparsity of α_{ij} coefficient
- Introduce binary variables s_{ij} to model existence of α_{ij}
- Add constraints $-Us_{ij} \leq \alpha_{ij} \leq Us_{ij}$

Sparse Coefficient Formulation

MINLP Model

$$\text{minimize} \quad \sum_{i=1}^{n} \left\{ y_i(2m-1) + (1-y_i)\left(2 \sum_{j=1, j\neq i}^{n} s_{ij} - 1\right)\right\} \tag{7a}$$

$$\text{subject to} \quad v_i = \sum_{j=1}^{n} \alpha_{ij} v_j \qquad i = 1, \ldots, n \tag{7b}$$

$$-Us_{ij} \leq \alpha_{ij} \leq Us_{ij} \qquad i,j = 1, \ldots, n \tag{7c}$$

$$-U(1-y_i) \leq \alpha_{ij} \leq U(1-y_i) \qquad i,j = 1, \ldots, n \tag{7d}$$

$$s_{ij} \leq y_j \qquad i,j = 1, \ldots, n \tag{7e}$$

$$y_i \in \{0,1\}, \quad s_{ij} \in \{0,1\} \qquad i,j = 1, \ldots, n$$

Sparse Coefficient Formulation

Equivalent MILP Model

$$\text{minimize} \quad 2m\sum_{i=1}^{n} y_i + 2\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}(s_{ij}-z_{ij}) - n \qquad (7a)$$

$$\text{subject to} \quad v_i = \sum_{j=1}^{n} \alpha_{ij} v_j \qquad i = 1,\dots,n \qquad (7b)$$

$$-Us_{ij} \leq \alpha_{ij} \leq Us_{ij} \qquad i,j = 1,\dots,n \qquad (7c)$$

$$-U(1-y_i) \leq \alpha_{ij} \leq U(1-y_i) \qquad i,j = 1,\dots,n,\ i \neq \qquad (7d)$$

$$z_{ij} \leq y_i, \quad z_{ij} \leq s_{ij}, \quad z_{ij} \geq y_i + s_{ij} - 1, \qquad i,j = 1,\dots,n \qquad (7e)$$

$$y_i \in \{0,1\}, \quad z_{ij} \in \{0,1\}, \quad s_{ij} \in \{0,1\} \qquad i,j = 1,\dots,n. \qquad (7f)$$

Results

Initial Formulation

- Initial formulation only had sparsity in the α_{ij}
- MINTO was not able to produce some optimal solutions
  - Report results after 36000 seconds

	Default	MILP			Sparse Coef. MILP		
Element	Flops	Flops	LPs	CPU	Flops	LPs	CPU
P_1 2D	42	42	33	0.10	34	187	0.43
P_2 2D	147	147	2577	37.12	67	6030501	36000
P_1 3D	170	166	79	0.49	146	727	3.97
P_2 3D	935	935	25283	36000	829	33200	36000

Results

Formulation with Sparse Basis

- We can also take account of the sparsity in the basis vectors
- Count only the flops for nonzero entries
  - Significantly decreases the flop count

Elements	Sparse Coefficient Flops	Sparse Basis Flops
P_1 2D	34	12
P_1 3D	146	26

Outline

Sieve

Sieve is an interface for

- general topologies
- functions over these topologies (bundles)
- traversals

One relation handles all hierarchy

- Vast reduction in complexity
    - Dimension independent code
    - A single communication routine to optimize
- Expansion of capabilities
    - Partitioning and distribution
    - Hybrid meshes
    - Complicated structures and embedded boundaries
    - Unstructured multigrid

FIAT

Finite Element Integrator And Tabulator by Rob Kirby

http://fenicsproject.org/

FIAT understands

- Reference element shapes (line, triangle, tetrahedron)
- Quadrature rules
- Polynomial spaces
- Functionals over polynomials (dual spaces)
- Derivatives

Can build arbitrary elements by specifying the Ciarlet triple (K, P, P')

FIAT is part of the FEniCS project

FIAT

Finite Element Integrator And Tabulator by Rob Kirby

http://fenicsproject.org/

FIAT understands

- Reference element shapes (line, triangle, tetrahedron)
- Quadrature rules
- Polynomial spaces
- Functionals over polynomials (dual spaces)
- Derivatives

Can build arbitrary elements by specifying the Ciarlet triple (K, P, P')

FIAT is part of the FEniCS project

Example: Discontinuous Galerkin Poisson

Poisson

$$-\Delta u = f \qquad \text{on} \qquad \Omega = [0,1] \times [0,1]$$

Using a discontinuous Galerkin formulation (interior penalty method).

- Define our Form and compile (FIAT + FFC)
- Define our Simulation (DOLFIN)
  - Define our mesh
  - Assemble and solve
  - Post process (visualize, error, ...)

Example: Discontinuous Galerkin Poisson

Defining the form

```
element = FiniteElement("Discontinuous Lagrange",
                        "triangle", 1)
...
n = FacetNormal("triangle")
h = MeshSize("triangle")
alpha = 4.0; gamma = 8.0
a = dot(grad(v), grad(u))*dx
  - dot(avg(grad(v)), jump(u, n))*dS
  - dot(jump(v, n), avg(grad(u)))*dS
  + alpha/h('+')*dot(jump(v, n), jump(u, n))*dS
  - dot(grad(v), mult(u, n))*ds
  - dot(mult(v, n), grad(u))*ds + gamma/h*v*u*ds
```

see `ffc/src/demo/PoissonDG.form`, and compile with

```
$ ffc PoissonDG.form
```

Example: Discontinuous Galerkin Poisson

Writing the Simulation: Assemble and solve

```
// Create user defined functions
Source f(mesh); Flux g(mesh);
FacetNormal n(mesh);
AvgMeshSize h(mesh);
// Define PDE
PoissonBilinearForm a;
PoissonLinearForm    L(f, g);
LinearPDE            pde(a, L, mesh, bc);
// Solve PDE
Function u;
pde.solve(u);
```

Example: Discontinuous Galerkin Poisson

Simulate!

Outline

Conclusions

Better mathematical abstractions
bring concrete benefits

- Vast reduction in complexity
  - Declarative, rather than imperative, specification
  - Dimension independent code
- Opportunites for optimization
  - Higher level operations missed by traditional compilers
  - Single communication routine to optimize
- Expansion of capabilities
  - Easy model definition
  - Arbitrary elements
  - Complex geometries and embedded boundaries

Outline

Hierarchy Abstractions

- Generalize to a set of linear spaces
  - `Sieve` provides topology, can also model `Mat`
  - `Section` generalizes `Vec`
  - Spaces interact through an `Overlap` (just a `Sieve`)
- Basic operations
  - Restriction to finer subspaces, `restrict()`/`update()`
  - Assembly to the subdomain, `complete()`
- Allow reuse of geometric and multilevel algorithms

Unstructured Interface (before)

- Explicit references to element type
    - getVertices(edgeID), getVertices(faceID)
    - getAdjacency(edgeID, VERTEX)
    - getAdjacency(edgeID, dim = 0)
- No interface for transitive closure
    - Awkward nested loops to handle different dimensions
- Have to recode for meshes with different
    - dimension
    - shapes

Unstructured Interface (before)

- Explicit references to element type
  - getVertices(edgeID), getVertices(faceID)
  - getAdjacency(edgeID, VERTEX)
  - getAdjacency(edgeID, dim = 0)
- No interface for transitive closure
  - Awkward nested loops to handle different dimensions
- Have to recode for meshes with different
  - dimension
  - shapes

Unstructured Interface (before)

- Explicit references to element type
  - getVertices(edgeID), getVertices(faceID)
  - getAdjacency(edgeID, VERTEX)
  - getAdjacency(edgeID, dim = 0)
- No interface for transitive closure
  - Awkward nested loops to handle different dimensions
- Have to recode for meshes with different
  - dimension
  - shapes

Go Back to the Math

Combinatorial Topology gives us a framework for geometric computing.

- Abstract to a relation, covering, on sieve points
  - Points can represent any mesh element
  - Covering can be thought of as adjacency
  - Relation can be expressed in a DAG (Hasse Diagram)
- Simple query set:
  - provides a general API for geometric algorithms
  - leads to simpler implementations
  - can be more easily optimized

Go Back to the Math

Combinatorial Topology gives us a framework for geometric computing.

- Abstract to a relation, covering, on sieve points
  - Points can represent any mesh element
  - Covering can be thought of as adjacency
  - Relation can be expressed in a DAG (Hasse Diagram)
- Simple query set:
  - provides a general API for geometric algorithms
  - leads to simpler implementations
  - can be more easily optimized

Go Back to the Math

Combinatorial Topology gives us a framework for geometric computing.

- Abstract to a relation, covering, on sieve points
  - Points can represent any mesh element
  - Covering can be thought of as adjacency
  - Relation can be expressed in a DAG (Hasse Diagram)
- Simple query set:
  - provides a general API for geometric algorithms
  - leads to simpler implementations
  - can be more easily optimized

Unstructured Interface (after)

- NO explicit references to element type
  - A point may be any mesh element
  - getCone(point): adjacent (d-1)-elements
  - getSupport(point): adjacent (d+1)-elements
- Transitive closure
  - closure(cell): The computational unit for FEM
- Algorithms independent of mesh
  - dimension
  - shape (even hybrid)
  - global topology
  - data layout

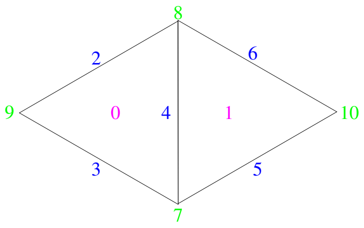
Unstructured Interface (after)

- NO explicit references to element type
  - A point may be any mesh element
  - getCone(point): adjacent (d-1)-elements
  - getSupport(point): adjacent (d+1)-elements
- Transitive closure
  - closure(cell): The computational unit for FEM
- Algorithms independent of mesh
  - dimension
  - shape (even hybrid)
  - global topology
  - data layout

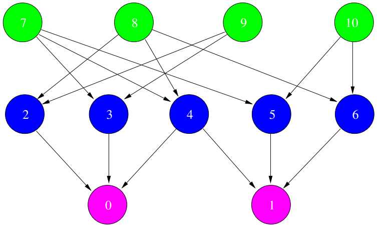
Unstructured Interface (after)

- NO explicit references to element type
  - A point may be any mesh element
  - getCone(point): adjacent (d-1)-elements
  - getSupport(point): adjacent (d+1)-elements
- Transitive closure
  - closure(cell): The computational unit for FEM
- Algorithms independent of mesh
  - dimension
  - shape (even hybrid)
  - global topology
  - data layout

Doublet Mesh

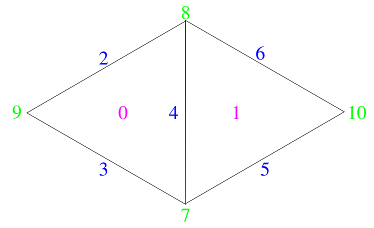


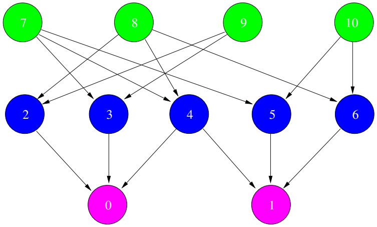
$$\begin{tikzcd}
7 \arrow[d] \arrow[rd] \arrow[rrd] & 8 \arrow[ld] \arrow[rd] \arrow[rrrd] & 9 \arrow[llld] \arrow[ld] & & 10 \arrow[ld] \arrow[d] \\
2 \arrow[rd] & 3 \arrow[d] & 4 \arrow[ld] \arrow[rd] & 5 \arrow[d] & 6 \arrow[ld] \\
 & 0 & & 1 &
\end{tikzcd}$$



- Incidence/covering arrows
- *cone*(0) = {2, 3, 4}
- *support*(7) = {2, 3}

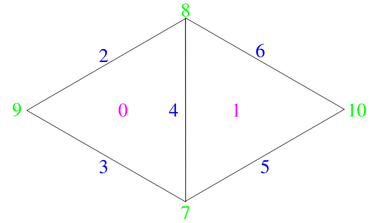
Doublet Mesh

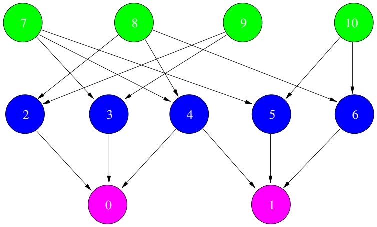




- Incidence/covering arrows
- $cone(0) = \{2, 3, 4\}$
- $support(7) = \{2, 3\}$

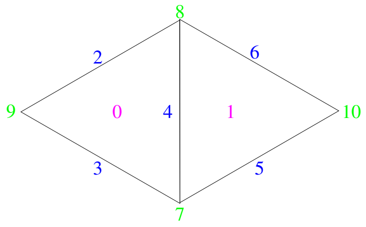
Doublet Mesh

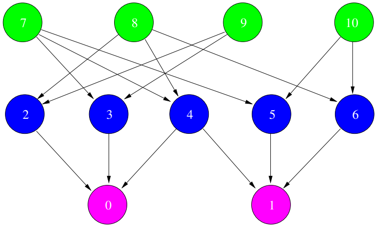




- Incidence/covering arrows
- *cone*(0) = {2, 3, 4}
- *support*(7) = {2, 3}

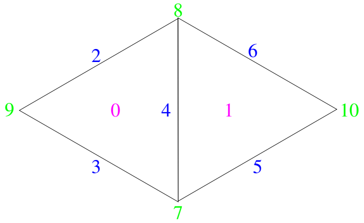
Doublet Mesh

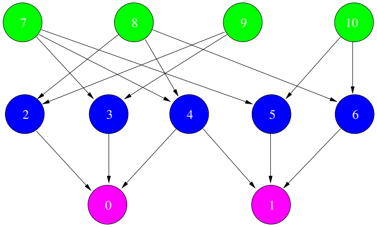




- Incidence/covering arrows
- $\textit{closure}(0) = \{0, 2, 3, 4, 7, 8, 9\}$
- $\textit{star}(7) = \{7, 2, 3, 0\}$

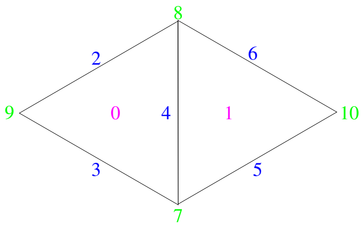
Doublet Mesh

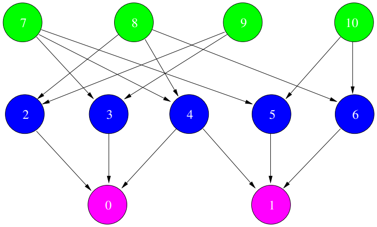




- Incidence/covering arrows
- *closure*(0) = {0, 2, 3, 4, 7, 8, 9}
- *star*(7) = {7, 2, 3, 0}

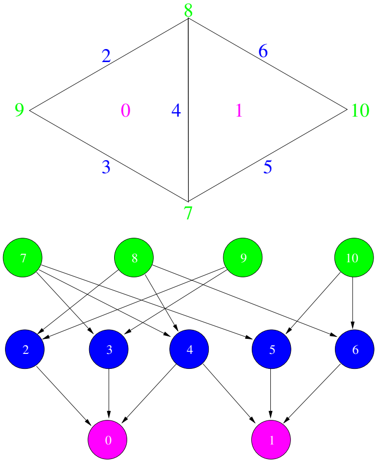
Doublet Mesh





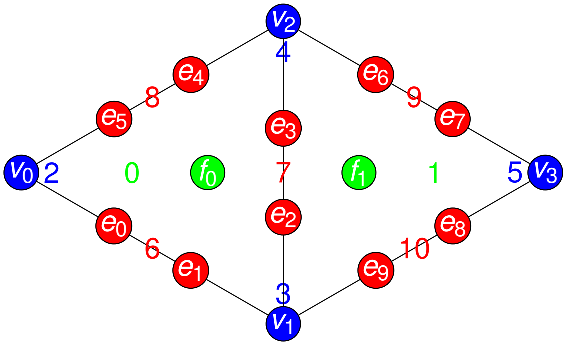
- Incidence/covering arrows
- *meet*$(0, 1) = \{4\}$
- *join*$(8, 9) = \{4\}$

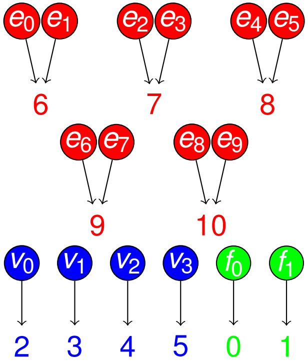
Doublet Mesh



- Incidence/covering arrows
- $\mathit{meet}(0, 1) = \{4\}$
- $\mathit{join}(8, 9) = \{4\}$

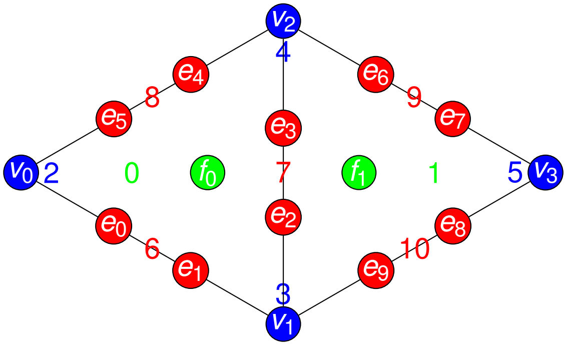
Doublet Section

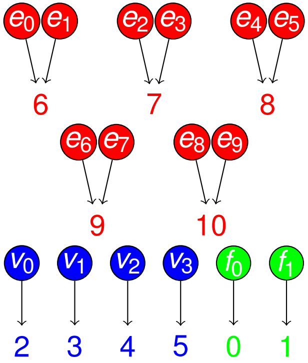




- **Section** interface
  - $restrict(0) = \{f_0\}$
  - $restrict(2) = \{v_0\}$
  - $restrict(6) = \{e_0, e_1\}$

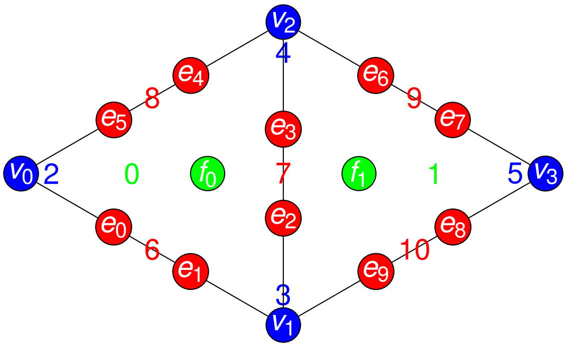
Doublet Section

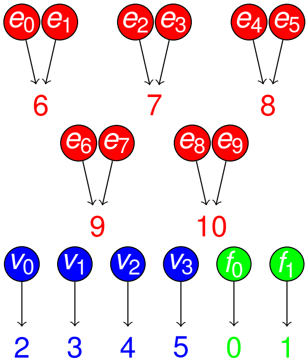




- **Section** interface
  - $restrict(0) = \{f_0\}$
  - $restrict(2) = \{v_0\}$
  - $restrict(6) = \{e_0, e_1\}$

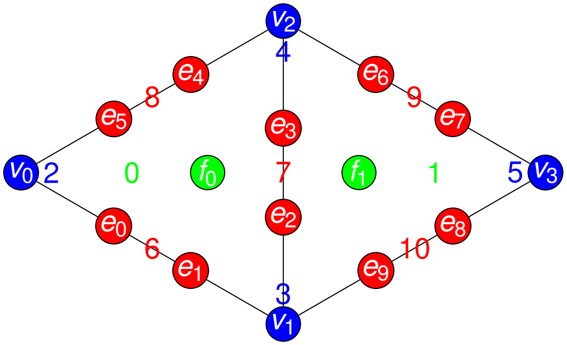
Doublet Section

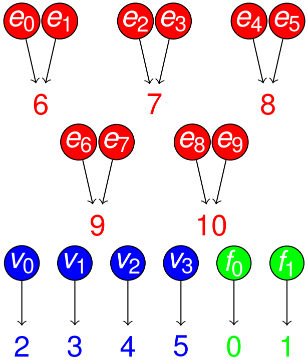




- **Section** interface
  - $restrict(0) = \{f_0\}$
  - $restrict(2) = \{v_0\}$
  - $restrict(6) = \{e_0, e_1\}$

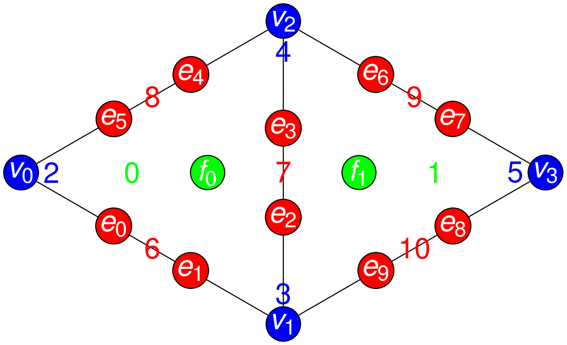
Doublet Section





- **Section** interface
  - $restrict(0) = \{f_0\}$
  - $restrict(2) = \{v_0\}$
  - $restrict(6) = \{e_0, e_1\}$

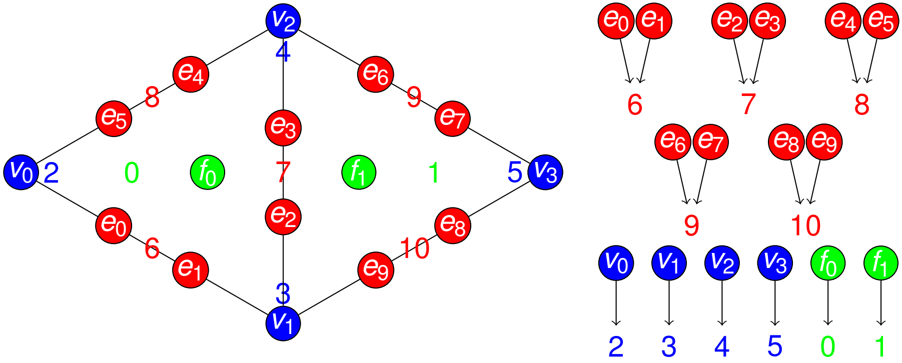
Doublet Section



$$\begin{tikzcd}
e_0 \arrow[d] & e_1 \arrow[dl] & e_2 \arrow[d] & e_3 \arrow[dl] & e_4 \arrow[d] & e_5 \arrow[dl] \\
6 & & 7 & & 8 & \\
e_6 \arrow[d] & e_7 \arrow[dl] & e_8 \arrow[d] & e_9 \arrow[dl] & & \\
9 & & 10 & & & \\
v_0 \arrow[d] & v_1 \arrow[d] & v_2 \arrow[d] & v_3 \arrow[d] & f_0 \arrow[d] & f_1 \arrow[d] \\
2 & 3 & 4 & 5 & 0 & 1
\end{tikzcd}$$

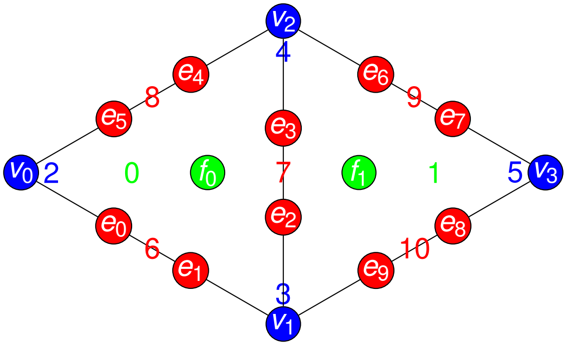
- Topological traversals: follow connectivity
  - *restrictClosure*$(0) = \{f_0\, e_0\, e_1\, e_2\, e_3\, e_4\, e_5\, v_0\, v_1\, v_2\}$
  - *restrictStar*$(7) = \{v_0\, e_0\, e_1\, e_4\, e_5\, f_0\}$

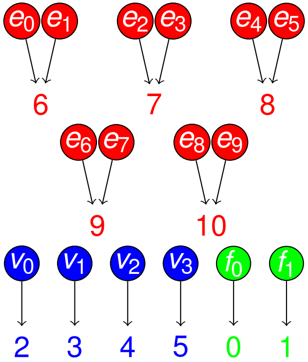
Doublet Section



- Topological traversals: follow connectivity
  - $\mathit{restrictClosure}(0) = \{f_0\, e_0\, e_1\, e_2\, e_3\, e_4\, e_5\, v_0\, v_1\, v_2\}$
  - $\mathit{restrictStar}(7) = \{v_0\, e_0\, e_1\, e_4\, e_5\, f_0\}$

Doublet Section





- Topological traversals: follow connectivity
  - $restrictClosure(0) = \{f_0\, e_0\, e_1\, e_2\, e_3\, e_4\, e_5\, v_0\, v_1\, v_2\}$
  - $restrictStar(7) = \{v_0\, e_0\, e_1\, e_4\, e_5\, f_0\}$

Global and Local

Local (analytical)

- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions
- Largely dim dependent (e.g. quadrature)

Global (topological)

- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies
- Largely dim independent (e.g. mesh traversal)

Global and Local

Local (analytical)

- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions
- Largely dim dependent (e.g. quadrature)

Global (topological)

- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies
- Largely dim independent (e.g. mesh traversal)

Global and Local

Local (analytical)

- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions
- Largely dim dependent (e.g. quadrature)

Global (topological)

- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies
- Largely dim independent (e.g. mesh traversal)

Global and Local

Local (analytical)

- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions
- Largely dim dependent (e.g. quadrature)

Global (topological)

- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies
- Largely dim independent (e.g. mesh traversal)

Global and Local

Local (analytical)

- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions
- Largely dim dependent (e.g. quadrature)

Global (topological)

- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies
- Largely dim independent (e.g. mesh traversal)

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  coords = mesh->restrict(coordinates, c);
  v0, J, invJ, detJ = computeGeometry(coords);
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  inputVec = mesh->restrict(U, c);
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    realCoords = J*refCoords[q] + v0;
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      elemVec[f] += basis[q,f]*rhsFunc(realCoords);
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      for(d = 0; d < dim; ++d)
        for(e) testDerReal[d] += invJ[e,d]*basisDer[q,
      for(g = 0; g < numBasisFuncs; ++g) {
        for(d = 0; d < dim; ++d)
          for(e) basisDerReal[d] += invJ[e,d]*basisDer
          elemMat[f,g] += testDerReal[d]*basisDerReal[
        elemVec[f] += elemMat[f,g]*inputVec[g];
      }
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      elemVec[f] += basis[q,f]*lambda*exp(inputVec[f])
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  mesh->updateAdd(F, c, elemVec);
}
<Aggregate updates>
```

Integration

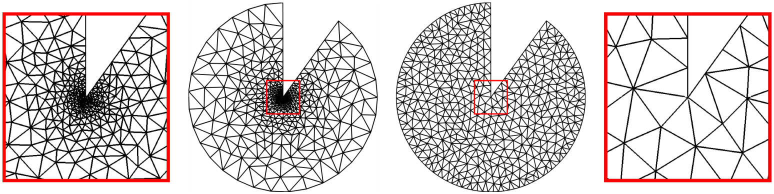
```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
<Aggregate updates>
```

Integration

```
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
  <Compute cell geometry>
  <Retrieve values from input vector>
  for(q = 0; q < numQuadPoints; ++q) {
    <Transform coordinates>
    for(f = 0; f < numBasisFuncs; ++f) {
      <Constant term>
      <Linear term>
      <Nonlinear term>
      elemVec[f] *= weight[q]*detJ;
    }
  }
  <Update output vector>
}
Distribution<Mesh>::completeSection(mesh, F);
```

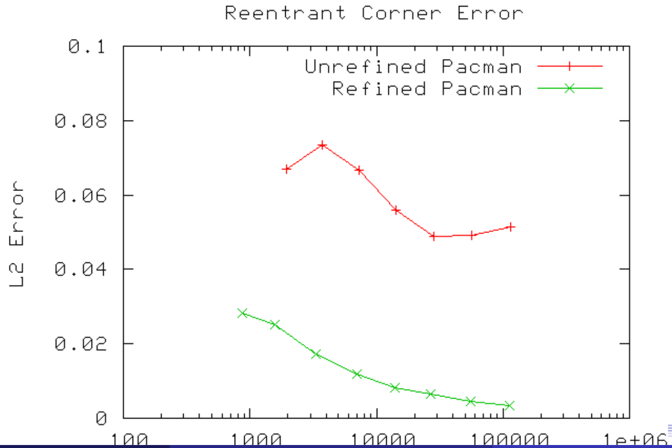
Reentrant Problems

- Reentrant corners need nonnuiform refinement to maintain accuracy
- Coarsening preserves accuracy in MG without user intervention



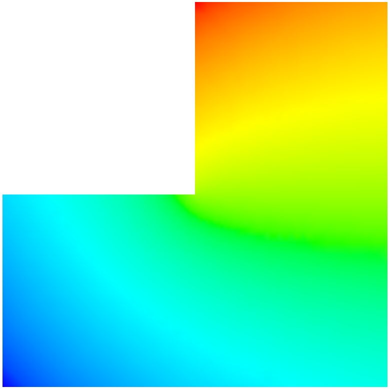
Reentrant Problems

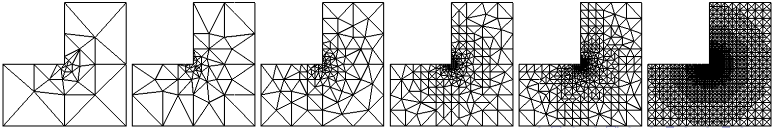
- Reentrant corners need nonnuiform refinement to maintain accuracy
- Coarsening preserves accuracy in MG without user intervention



Reentrant Problems

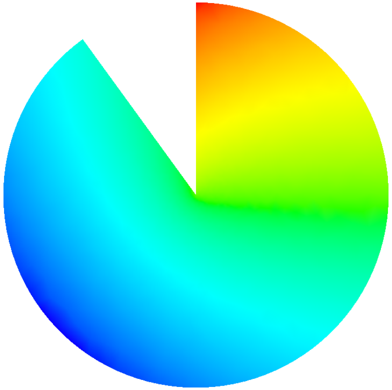
Exact Solution for reentrant problem: $u(x, y) = r^{\frac{2}{3}} sin(\frac{2}{3}\theta)$

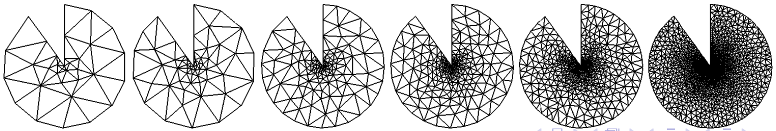




Reentrant Problems

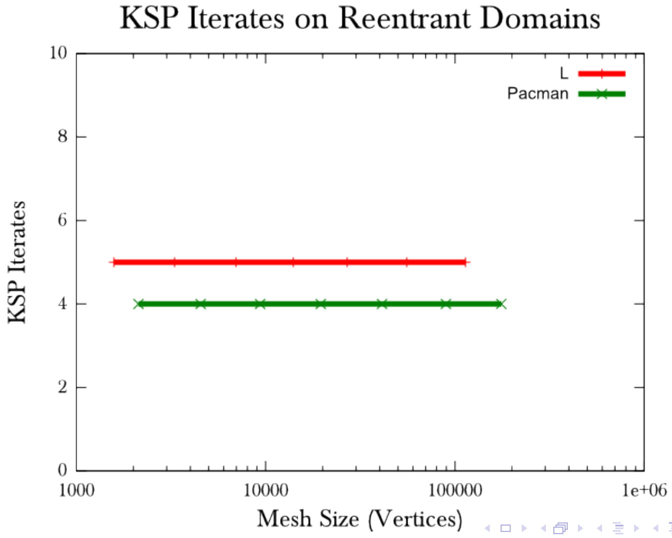
Exact Solution for reentrant problem: $u(x, y) = r^{\frac{2}{3}} sin(\frac{2}{3}\theta)$





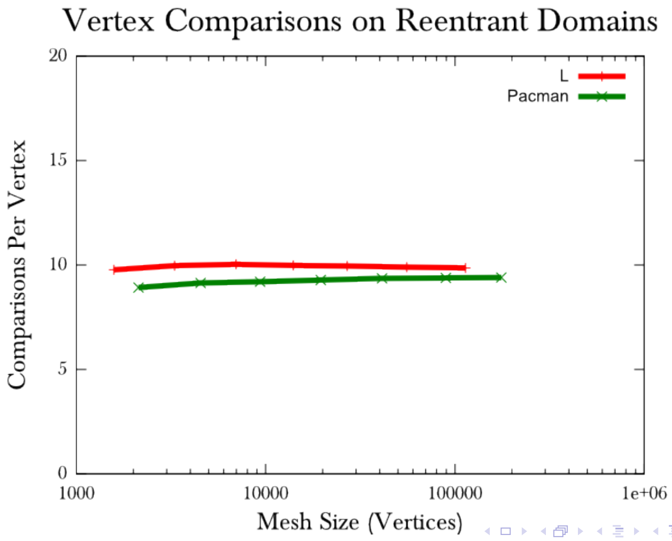
GMG Performance

Linear solver iterates are constant as system size increases:



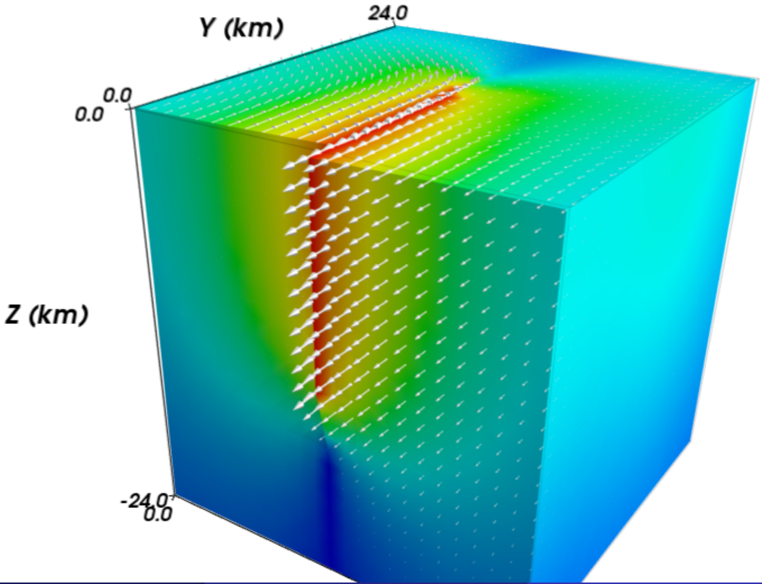
GMG Performance

Work to build the preconditioner is constant as system size increases:



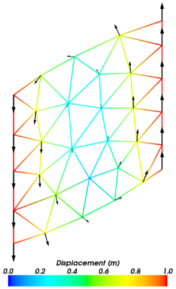
Outline

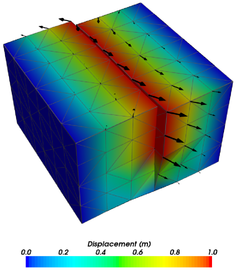
Reverse-slip Benchmark



Multiple Mesh Types

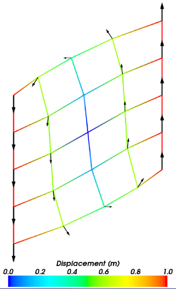
Triangular

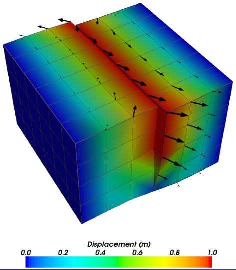




Tetrahedral

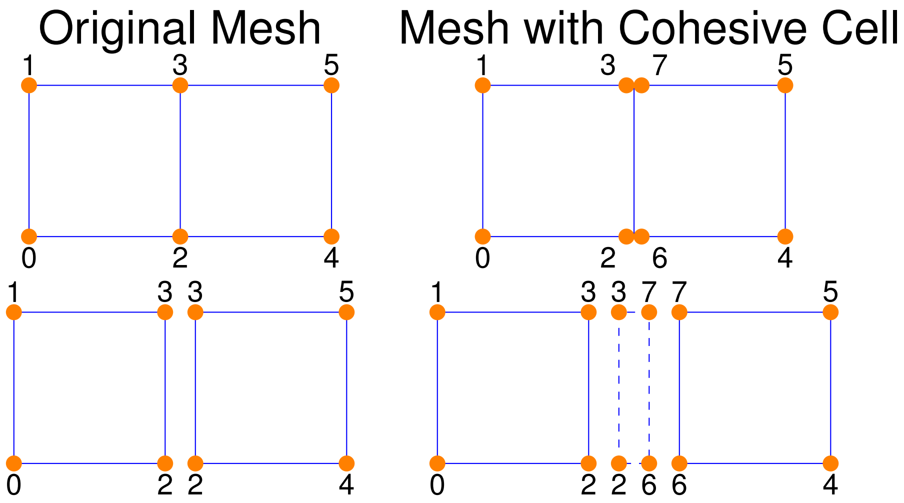
Rectangular





Hexahedral

Cohesive Cells



Exploded view of meshes

Cohesive Cells

Cohesive cells are used to enforce slip conditions on a fault

- Demand complex mesh manipulation
  - We allow specification of only fault vertices
  - Must “sew” together on output
- Use Lagrange multipliers to enforce constraints
  - Forces illuminate physics
- Allow different fault constitutive models
  - Simplest is enforced slip
  - Now have fault constitutive models

Partial Geometry

Given a set V and a set of lines $L \subset \mathcal{P}(V)$, (V, L) is a partial geometry if

- there is at most one line through each pair of points
- each line has at least three point

Note that

- Typical geometries have exactly one line through each pair of points
- Encoded by ternary relations, like coplanarity, which satisfy

$$R(x, y, z) \wedge R(y, z, p) \Rightarrow R(x, y, p) \wedge R(y, z, p)$$

- Generalizes to higher arity relations

Definitions

- **vertex**
  - A point lying in two or more lines
- **closure**
  - The transitive closure $\bar{S}$ under R of some $S \subset V$
  - $z \in V \wedge \exists x, y \in S \ni R(x, y, z) \Rightarrow z \in \bar{S}$
- **independent set**
  - A set S such that for any $S' \subset S$, $\bar{S}' \neq S$
- **basis**
  - An independent set S such that $\bar{S} = V$

Goal

We want a basis of minimal cost, which now means size.

- Something like a “minimum spanning hypertree”
- Closure operation produces a DAG
  - Use topological sort to get computation sequence
- Complexity is unknown
- Unfortunate example shows bases of differing size
  - At odds with matroid theory

Geometric Reduction

- Eliminate parallel lines (no vertices)
  - Can add any two points on the line to a minimal basis
- Eliminate single vertex lines
  - Can add any non-vertex on the line to a minimal basis
- Eliminate non-vertices from basis
  - Each line has at least two vertices
    - If two vertices are already present, discard point
    - Otherwise, switch with a vertex
  - The generated set is the same, and the size has not increased

Exchange Property

We want to show that all reduced bases are the same size.

- Remove a vertex p from the basis B
  - Now there is a set $Ex(p)$ which is no longer in $\bar{B}$
- Choose q from $Ex(p)$
- Reverse the generation path from p to q
  - If we generate p, we generate all of $Ex(p)$

Now

- We have an easy algorithm for a minimal basis
- Matroid results apply