Mesquite = Mesh Quality Improvement Toolkit

A C++ software library to improve mesh quality by vertex-movement.

Purpose is to support
1. *a priori* mesh generation,
2. accuracy & efficiency of PDE simulations using *a posteriori* information to improve the mesh,
3. mesh optimization research.

A SciDAC ITAPS Tool

http://www.cs.sandia.gov/optimization/knupp/Mesquite.html

mesquite@software.sandia.gov

Patrick Knupp, Lori Diachin, Jason Kraftcheck, Brian Miller, Martin Isenburg

(Only general purpose mesh improvement tool of its kind)

SAND2009-7258P

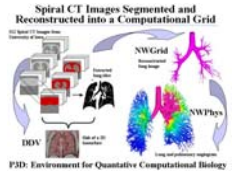
Mesquite Applications

- UIUC Rocket Center Propellant Burn (shape improvement)

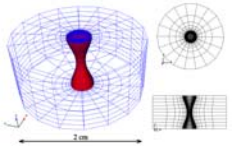
- SLAC Waveguide cavity shape design optimization (deforming meshes)

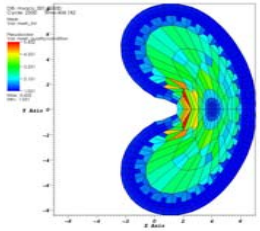
- Cubit (a priori mesh improvement)

- SkiRun Project (medical)

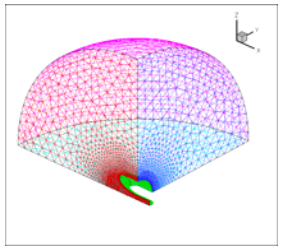


- PNNL NWGrid (improvement of biological meshes)
- Alegra and ALE3D (Invertibility guarantee for ALE rezone)





- NASA-Langley (sliver removal for viscous CFD mesh)



- Kull (research improved ALE rezone algorithms)
- PPL Fusion (mesh alignment and improvement)

Mesquite Capabilities Apply to Wide Variety of Mesh Types

Optimize the Quality of:

* 2D/3D Meshes composed of Triangles, Quadrilaterals, Tetrahedra, Hexahedra, Wedges, Pyramids,
* Unstructured, Structured, Hybrid meshes,
* Finite element meshes with high-order nodes

with

* Invertibility Guarantees, and
* Mesh untangling.

Technical Approach

Optimization-based vertex-movement

Find a local minimum of $F(x_1,\ldots,x_N)=\frac{1}{K}\sum_k\left(c_k\mu_k\right)^p$

Local quality at a mesh sample point (or element) given by μ.

Target-matrix Paradigm (state-of-the-art):

Optimal mesh defined by target-matrices representing ideal Jacobian matrix at each sample point. Only uses 3 basic quality metrics. Targets are automatically constructed using a priori data available to the application.

* Permits rapid deployment of custom-made optimizers.

Mesquite also has relaxation-smoothers (e.g. Laplace, constrained Laplace).

Notes

Input:

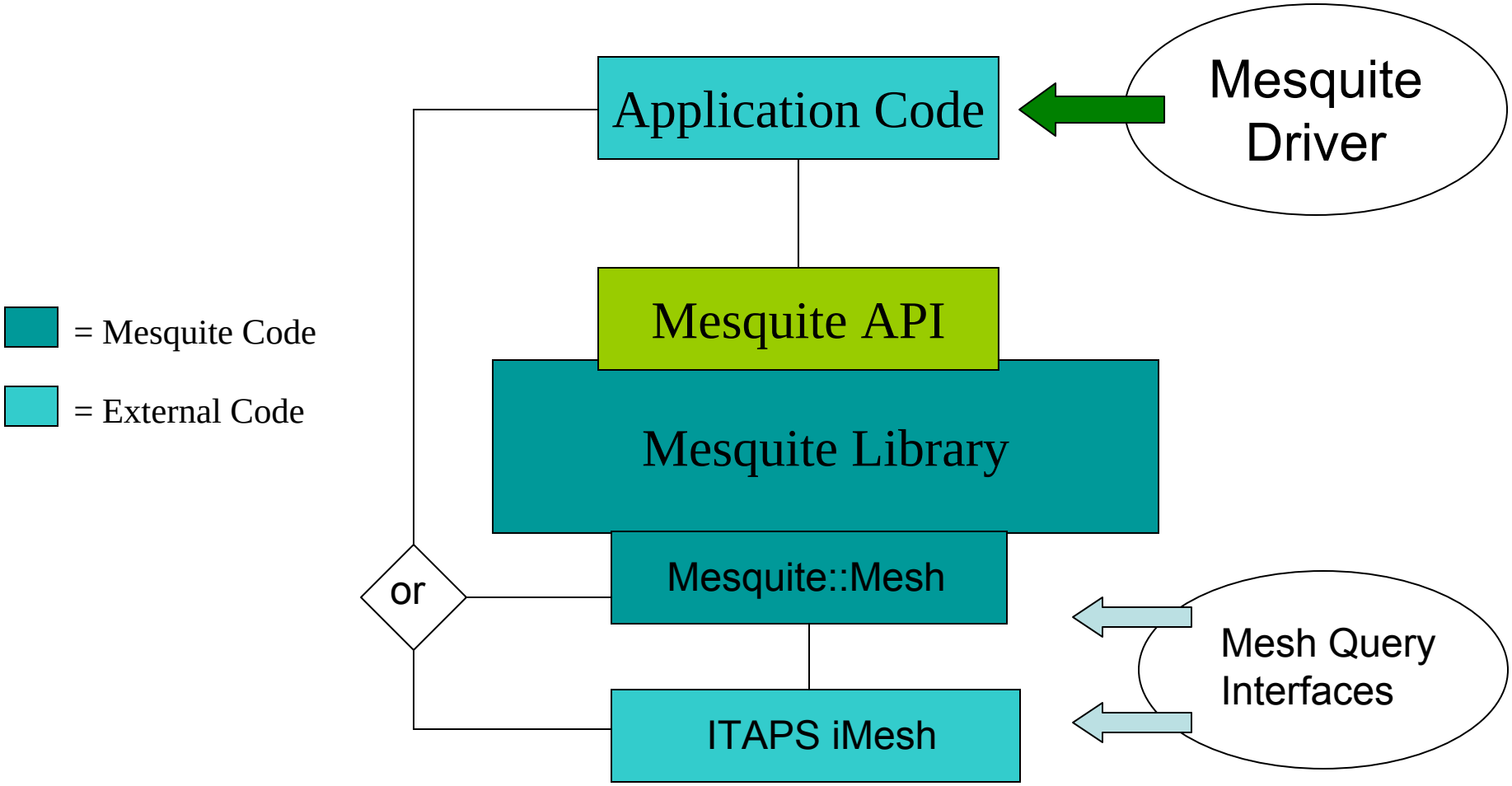
1. An initial mesh to be improved (vertices, connectivity, vertex constraints),
2. Termination tolerances and (rarely) a few tunable parameters,
3. Optional application data (scalar, vector, tensor fields).

Mesquite never halts execution if there is an error it can't deal with… returns control back to app.

Doubles as both an application service and a research platform
(so need to balance flexibility vs. efficiency)

Mesquite is limited to node-movement, but compliments other interoperable ITAPS tools: (e.g., swap, refine, geometry, visualization)

Mesquite Application Components



Mesquite API

Multi-level:

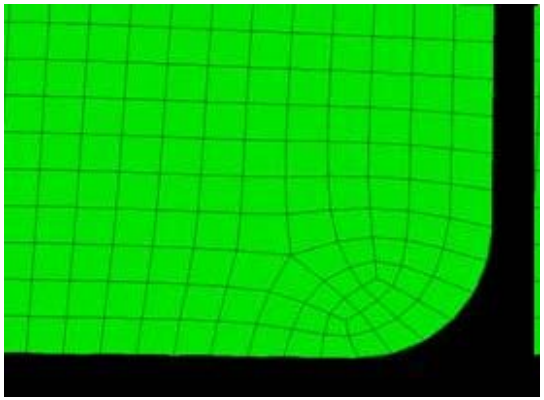
* low-level for custom-built smoothers (developer oriented),
* high-level for canned algorithms, instructions, and wrappers (user-oriented)

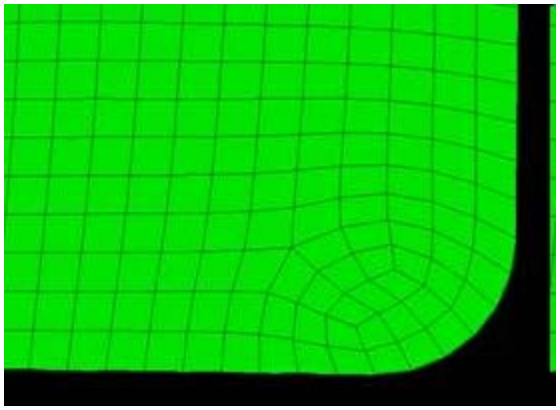
Current Mesquite Wrappers:

1. (smart) Laplacian Smoothing
2. Shape Improvement
3. Min-Edge Length Improvement
4. Shape Improvement for Size-adapted mesh
5. Sliver removal for CFD viscous boundary layer tet meshes
6. Synchronous Parallel Shape Improvement Wrapper

```
int main( int argc, char* argv[] ) {

    MsqError err;
    Mesquite:: MeshImpl my_mesh;
    my_mesh.read_vtk(argv[1], err);

    Mesquite::ShapeImprovementWrapper mesh_quality_algorithm(err);
    mesh_quality_algorithm.run_instructions( &my_mesh, err );

    return 0;
}
```

Minimum Edge-Length Improvement Wrapper

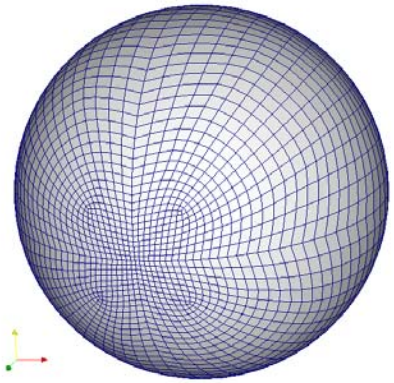


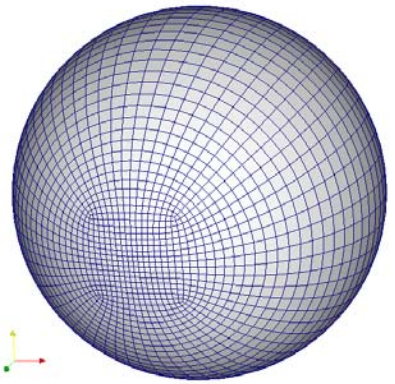


before

after

Size-adapted Mesh Shape Improvement Wrapper

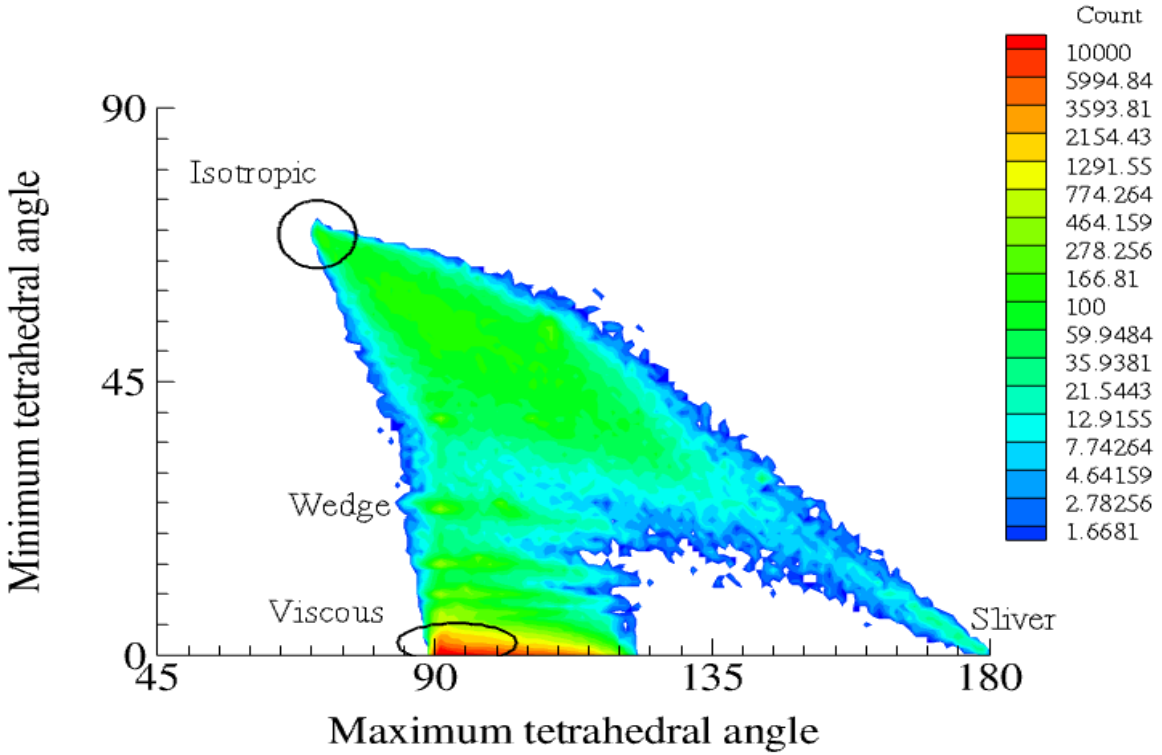




Optimizing the Quality of a tetrahedral viscous CFD mesh

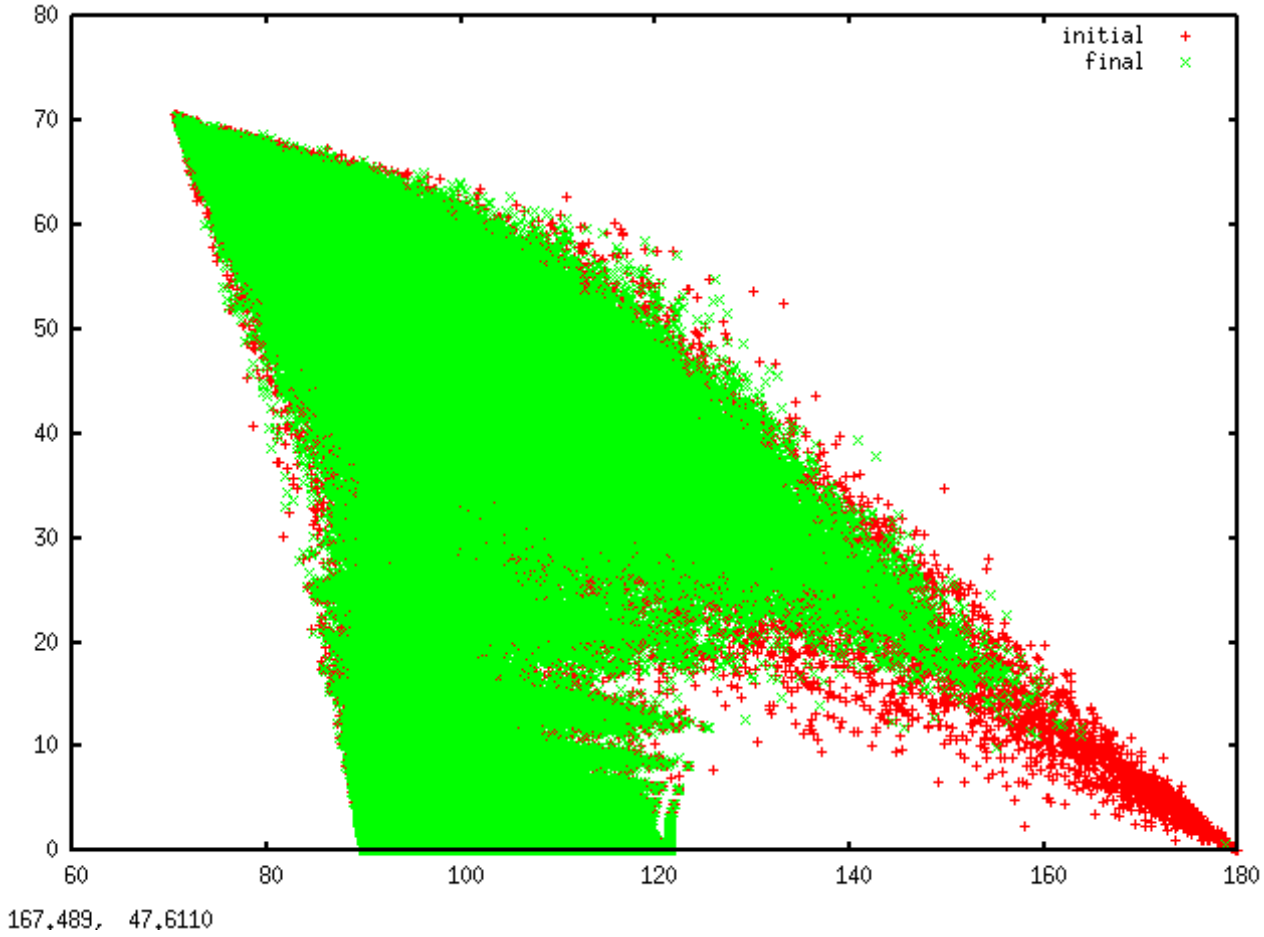
Test Mesh: Total elements 537975, Total nodes 97144.
Total viscous nodes 86610.

#elements vs. min & max dihedral angle

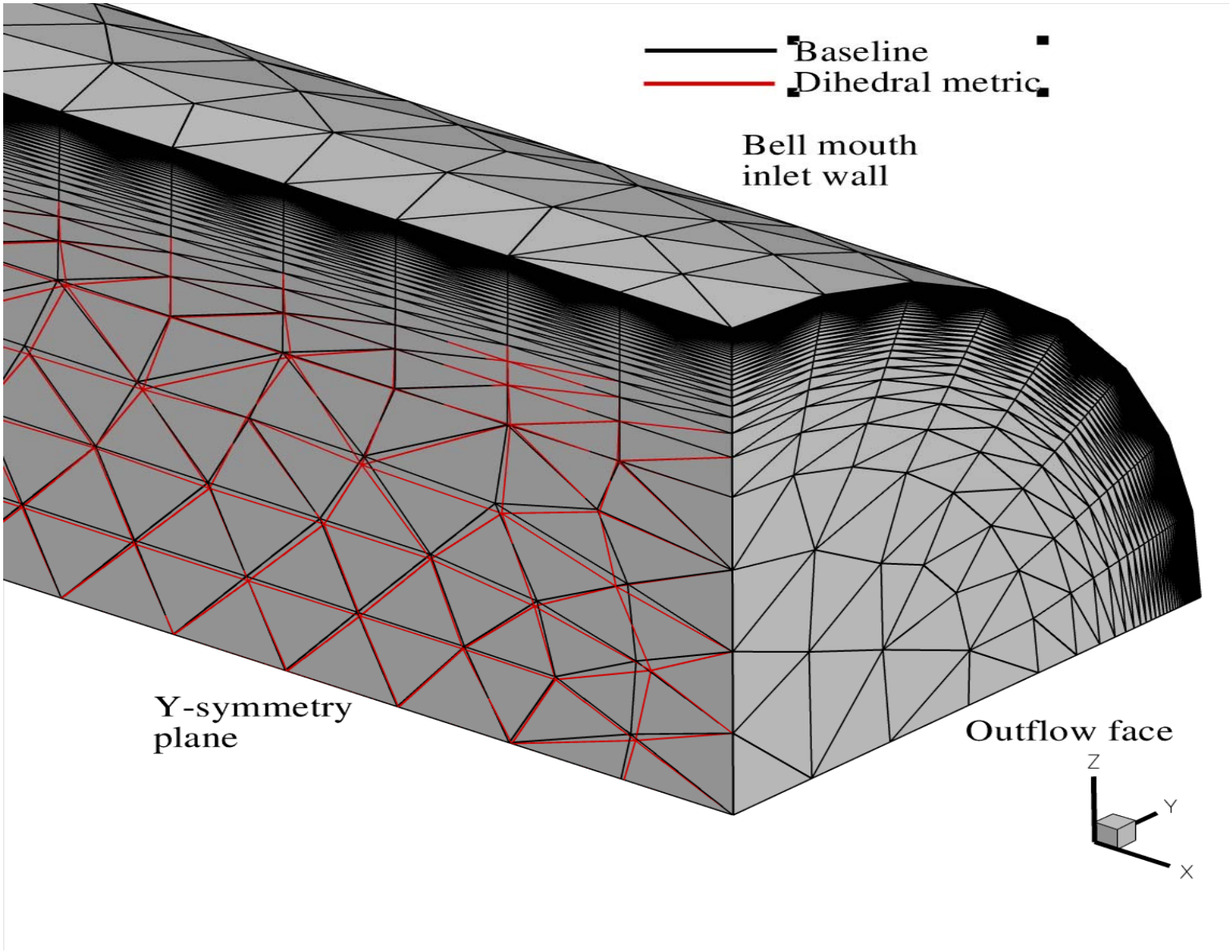


Results

Number of Sliver Elements Considerably Reduced
Viscous & Transition Regions seem Unaffected



Size Transition is Preserved



Synchronous Parallel Shape Improvement Wrapper

- Algorithm from Freitag, Jones & Plassman (1999)
- Weak scalability tests up to 128 processors successful
- Parallel result same as serial
- Works for most linear element types
- Requirements
  - one layer of ghost nodes
  - two tags for all nodes: global ID + processor ID
  - MPI

Using Parallel Mesquite (with ITAPS)

```
MPI_Init(&argc, &argv);
// load the ITAPS mesh
iMesh_getRootSet(mesh, &root_set, &err);
iMeshP_createPartitionAll(mesh, MPI_COMM_WORLD, &partition, &err);
iMeshP_loadAll(mesh, partition, root_set, iname, 0, &err, strlen(iname), 0);
// create ghost entities
iMeshP_createGhostEntsAll(mesh, partition, 3, 1, 1, 0, &err);
msqiMeshP = Mesquite::MsqIMeshP::create(mesh, root_set, MPI_COMM_WORLD,
                                        iBase_REGION, merr, fixed_tag_name);
alg->use_global_patch();
termInner = new Mesquite::TerminationCriterion();
termInner->add_criterion_type_with_double(Mesquite::TerminationCriterion::GRADIENT_L2_NORM_ABSOLUTE,1e-3,merr);
termInner->add_criterion_type_with_int(Mesquite::TerminationCriterion::NUMBER_OF_ITERATES,5,merr);
termOuter = new Mesquite::TerminationCriterion();
// run the instruction queue
// delete ghost entities
// cleanup
delete msqiMeshP;
iMeshP_destroyPartitionAll(mesh, partition, &err);
iMesh_dtor(mesh, &err);
MPI_Finalize();
```

Low-Level API

Access C++ Objects for custom-built mesh optimization:

* Qualtiy Metric
  - Element, Vertex, or Sample point
  - Composite (add, subtract, scale)
  - Analytic gradients and Hessians, Numerical

* Objective Function
  - L2, L-infinity, Power-mean templates
  - Composite (add, subtract, scale)

* QualityImprover
  - Feasible Newton, Quasi-Newton, Trust Region (Hessian-based), Conjugate Gradient, and Steepest Descent
  - Global, Block Coordinate Descent, and Nash options,

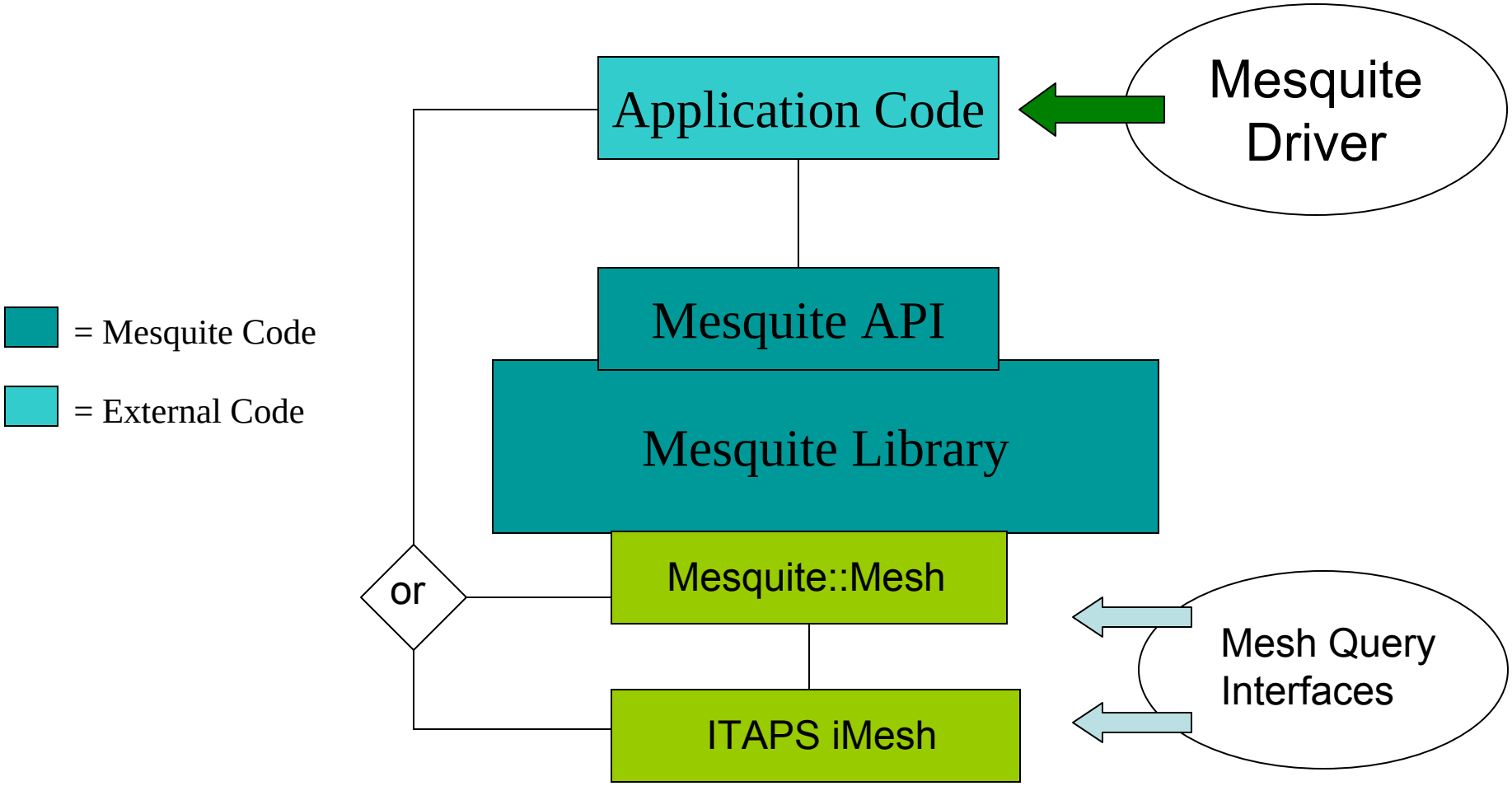
* Termination Criterion
  - Seven absolute and seven relative criteria,
  - can be compounded (OR, AND)

* Instruction Queue for sequential and multi-stage optimization procedures

* QualityAssessor for evaluating “before” & “after” quality

*TargetCalculator for creating targets in Target-matrix paradigm

Mesquite Application Components



Mesquite::Mesh Interface and the ITAPS iMesh Interface

Internal C++ mesh API on top of ITAPS iMesh
- Mesquite specific (more efficient)
- Fewer functions to implement
- Less general
- Stable

ArrayMesh class to help interface with Fortran code
VTK 3.0 mesh file reader/writer (with stand-alone Exodus to VTK converters)
Tag Data to access data stored with application (scalars, vectors, matrices)

ITAPS iMesh mesh interface
-Permits interoperability with other ITAPS tools via standard
-Can implement in C or Fortran

Geometry and Constraints:
- Mesquite has no general internal geometry engine,
- PlanarDomain, SphereDomain, CylinderDomain
- Has call-back functions “get_normal” and “move_to_owner” to constrain vertex movement to geometry
- No constrained or parametric optimization
- Fixed, Free, Slave flags

General Information

* Version 2.1 released October 5, 2009.

* Available from
  - MSQ web-page,
  - ITAPS code repository (RPI),
  - Trilinos (beta repository),
  - Cubit (only some capabilities exposed).

* Open source with LGPL license (downloaded 775 times over last 5 years)

• Compiles under GNU automake/libtool or cmake build systems

• Trilinos cmake/ctest

• Platforms: MSWindows, Unix

* Can compile as shared library

* No other libraries needed to build

* Extensive test suite (end-to-end and unit) w/nightly regression

* Users Guide (needs work)

Future Work

New Mesquite Wrappers (via Target-matrix Paradigm):

1. Skew Improvement
2. Updating Meshes on Deforming Domains
3. Shape Improvement with Size-Equidistribution
4. Anisotropic mesh smoothing
5. Heterogeneously-sized mesh smoothing
6. Geometric adaptivity
7. R-adaptivity to discretization error
8. Mesh alignment
9. Boundary mesh controller (skew, spacing)
10. Smoothing of Transition Layers
11. High-order node Smoothing

Additional Capabilities:

1. Smooth meshes with hanging nodes
2. Smooth dual mesh (Voronoi elements)
3. Node re-ordering for Efficiency
4. Relaxation solvers
5. ActiveSetSolver (restored)
6. Vertex Culling Methods
7. Mesh Untanglers