THE MESQUITE MESH QUALITY IMPROVEMENT TOOLKIT

Michael Brewer[1] Lori Freitag Diachin[1] Patrick Knupp[1] Thomas Leurent[2]
Darryl Melander[1]

[1]*Sandia National Laboratories, Albuquerque, NM U.S.A*
{mbrewer,ladiach,pknupp,djmelan}@sandia.gov
[2]*Argonne National Laboratories, Chicago, IL U.S.A. tleurent@mcs.anl.gov*

ABSTRACT

We describe the design goals and architecture of the Mesquite toolkit, a stand-alone library consisting of state-of-the-art algorithms for mesh quality improvement. The primary considerations in Mesquite design are to ensure that it is comprehensive, effective, efficient, and extensible. We give an overview of the Mesquite architecture and highlight the core classes, their inter-relations, and functionality. We describe the interfaces developed to obtain information from the mesh and geometry and to provide user-control of the quality metrics, algorithms, and termination criterion. Smoothing results for several meshes with a broad range of characteristics are given which showcase Mesquite's versatility.

Keywords: mesh quality, mesh improvement, mesh smoothing, mesh generation

1. INTRODUCTION

Mesh quality is critical for accuracy and efficiency in the solution of PDE-based applications. Quality can be affected in many stages of the solution process from de-featuring CAD models, to mesh generation procedures and $h-$ and $r-$adaptive schemes. At any stage after a mesh is created, its quality can be improved by node point movement methods, commonly called mesh smoothing, and topology modification schemes such as edge and face flipping. There are a number of techniques that have been developed for mesh improvement ranging from simple Laplacian smoothing [1] to more sophisticated algorithms such as Winslow smoothers for structured meshes [2] and the numerical optimization methods and topology modification schemes recently developed for unstructured meshes (for example, [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]).

The software associated with many of these references is largely contained in larger mesh generation or application frameworks. While many meshing codes such as ICEM-CFD can perform mesh improvement, there currently exist few stand-alone, portable libraries for mesh quality improvement that can be easily linked via functional interfaces to many different mesh generation and application codes. One such libary is Opt-MS [13], a stand-alone package that uses state-of-the-art optimization algorithms to improve homogeneous simplicial meshes. However, Opt-MS only provides patch-based smoothing algorithms for isotropic elements in volumes or on planar surfaces. No topological changes are supported. Under the auspices of the Tera-scale Simulation Tools and Technologies (TSTT) Center [14], we have recently begun development of a new toolkit, called Mesquite (Mesh Quality Improvement Toolkit), that is much more comprehensive than Opt-MS and, in this paper, we present our progress to date.

Successful development of Mesquite will provide a robust and effective mesh improvement tool to the broader scientific community. This will allow both mesh generation researchers and application scientists to benefit from the latest developments in mesh quality control and improvement. For example, Mesquite has already been used to smooth Geodesic meshes for the climate group at Colorado State University [15] and is presently being used to improve the quality of meshes used in the design of particle accelerators [16]. Preparations are also underway to link Mesquite to various TSTT meshing codes including CUBIT [17], Overture [18], and NWGrid [19].

The Mesquite design goals were first introduced in [20], and in this paper we compare the current status of Mesquite with that vision (Section 2), give an overview of the Mesquite design and architecture including the application programming interfaces (Sections 3 and 4), and show several examples of meshes with a broad range of characteristics that have been improved by Mesquite (Section 5).

2. MESQUITE DESIGN GOALS

Mesquite design goals are derived from a mathematical framework and are focused on providing a versatile, comprehensive, effective, inter-operable, and efficient library of mesh quality improvement algorithms that can be used by the non-expert and extended and customized by experts. In this section we highlight the current status of Mesquite in several of our design goal areas.

Mathematical Framework. Mesquite design is based on a currently evolving, but rigorous mathematical framework that poses the mesh quality improvement problem as an optimization problem. That is, suppose a mesh M contains n vertices, v_i, and k elements, e_i. Then the quality of each element or vertex is given as a general function $q_i(\mathbf{x})$ of the coordinates of the vertex locations. The index i ranges from 1 to n or 1 to k depending on whether a vertex-based or element-based metric is chosen. A mesh quality objective function $\mathcal{F} = f(q_i(\mathbf{x}))$ for $i = 1, \cdots, n_s$ or $i = 1, \cdots, k_s$ is formed to give an overall measure of mesh quality where n_s and k_s are the number of vertices or elements in a mesh sub-domain needing improvement, respectively. Mesquite is designed to solve the minimization problem min $\mathcal{F}$ for a broad collection of quality metrics q_i and objective functions $\mathcal{F}$.

Versatile and Comprehensive. Mesquite works on structured, unstructured, and hybrid meshes in both two and three dimensions. The design permits improvements to meshes composed of triangular, tetrahedral, quadrilateral, and hexahedral elements. Prismatic, pyramidal, and polyhedral elements can be easily added. It currently incorporates only methods for node movement; plans for topology modification and hybrid improvement strategies lie in the future. Node movement strategies include both local patch-based iteration schemes for one or a few free vertices and global methods which improve all vertices simultaneously. Mesquite will be applicable to both adaptive and nonadaptive meshing and to both low- and high-order discretization schemes, but currently works with non-adaptive meshes containing linear elements.

Effective. Mesquite uses state-of-the-art algorithms and metrics to guarantee improvement in mesh quality. Because the definition of mesh quality is application specific, we provide quality metrics that allow the user to untangle meshes, improve mesh smoothness, element size, and shape. In the future these metrics will be referenced to permit non-isotropic smoothing and adaptivity. The software is easily customizable, enabling users to insert their own quality metrics, objective functions, and algorithms and also provides mechanisms for creating combined approaches that use one or more improvement algorithms.

Inter-operable. To ensure that Mesquite is inter-operable with a large number of mesh generation packages, we will use the common interfaces for mesh query currently under development by the TSTT center. These interfaces provide uniform access to mesh geometry and topology and will be implemented by all TSTT center software including several DOE-supported mesh generation packages. We are working with the TSTT interface design team to ensure that Mesquite has efficient access to mesh and geometry information through strategies such as information caching and agglomeration. We are also participating in the design of interfaces needed to support topological changes generated by mesh swapping and flipping algorithms and to constrain vertices to the surface of a geometrical model.

Efficient. The outer layers of Mesquite use object-oriented design in C++ while the inner kernels use optimizable coding constructs such as arrays and inlined functions. To ensure efficient use of computationally intensive optimization algorithms, we employ inexpensive smoothers, such as Laplacian smoothing, as "preconditioners" for the more expensive optimization techniques. In addition, mesh culling algorithms can be used to smooth only those areas of the mesh that require improvement. Considerable attention has been devoted to understanding and implementing a variety of termination criteria that can be used to control the computational cost of the optimization algorithms.

3. MESQUITE ARCHITECTURE

The Mesquite architecture, shown in Figure 1, closely follows the abstractions defined by the mathematical framework to describe the optimization problem. In particular, the core abstract classes needed to define a mesh quality improvement algorithm

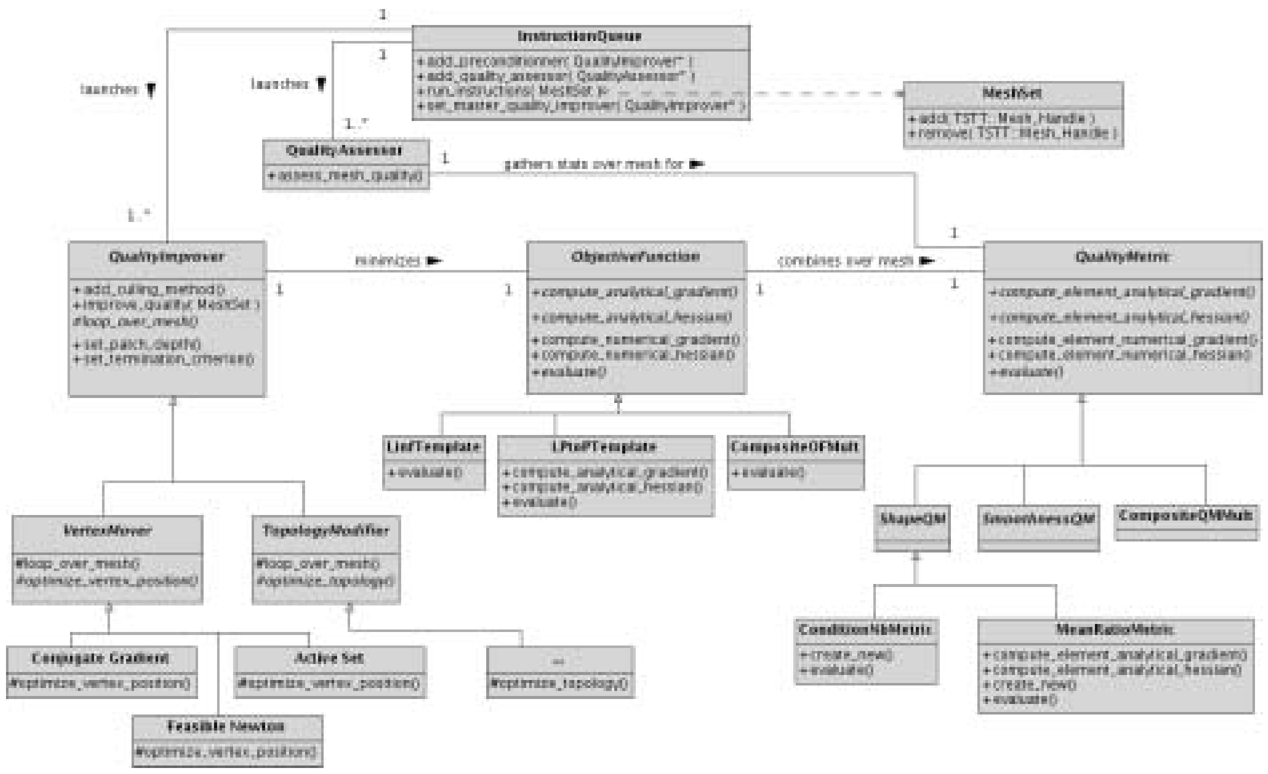

InstructionQueue
+add_preconditioner(QualityImprover*)
+add_quality_assessor(QualityAssessor*)
+run_instructions(MeshSet)
+set_master_quality_improver(QualityImprover*)
MeshSet
+add(TSTT::Mesh_Handle)
+remove(TSTT::Mesh_Handle)
launches
launches
1
1
1..*
1..*
QualityAssessor
+assess_mesh_quality()
1
gathers stats over mesh for
QualityImprover
+add_culling_method()
+improve_quality(MeshSet)
#loop_over_mesh()
+set_patch_depth()
+set_termination_criterion()
minimizes
1
1
ObjectiveFunction
+compute_analytical_gradient()
+compute_analytical_hessian()
+compute_numerical_gradient()
+compute_numerical_hessian()
+evaluate()
combines over mesh
1
1
1
QualityMetric
+compute_element_analytical_gradient()
+compute_element_analytical_hessian()
+compute_element_numerical_gradient()
+compute_element_numerical_hessian()
+evaluate()
LInfTemplate
+evaluate()
LPtoPTemplate
+compute_analytical_gradient()
+compute_analytical_hessian()
+evaluate()
CompositeOFMult
+evaluate()
ShapeQM
SmoothnessQM
CompositeQMMult
VertexMover
#loop_over_mesh()
#optimize_vertex_position()
TopologyModifier
#loop_over_mesh()
#optimize_topology()
ConditionNbMetric
+create_new()
+evaluate()
MeanRatioMetric
+compute_element_analytical_gradient()
+compute_element_analytical_hessian()
+create_new()
+evaluate()
Conjugate Gradient
#optimize_vertex_position()
Active Set
#optimize_vertex_position()
...
#optimize_topology()
Feasible Newton
#optimize_vertex_position()

Figure 1: Mesquite user interface UML class diagram. Abstract classes and virtual functions are in italic. Vertical links with a triangle indicate inheritance. Plain links indicate association. Protected functions are prefixed with '#', public ones with '+' .

are `QualityMetric`, `ObjectiveFunction` (which takes a `QualityMetric` as input), and `QualityImprover` (which takes an `ObjectiveFunction` as input).

In addition, a number of other classes have been created to support the needs of mesh quality improvement algorithms:

- `QualityAssessor`: to provide an evaluation of mesh quality using standard statistical procedures,
- `TerminationCriterion`: to customize the stopping criteria used with a mesh quality improvement algorithm,
- `InstructionQueue`: to compose quality improvers and quality assessors together to form efficient mesh quality improvement and evaluation methods, and
- `MeshSet` and `PatchData`: to provide the mechanisms for managing the application mesh and geometry information and the mesh sub-domains used in optimization procedures.

The Mesquite architecture uses as much dynamic polymorphism in the form of inheritance and virtual functions as is possible without degrading performance. This allows developers to easily add new functionality to Mesquite by inheriting from the appropriate abstract class and implementing its interface (i.e. its abstract virtual functions). For example, to implement a new quality metric, a user must inherit from the base `QualityMetric` class and implement only a single function that returns the value of the metric for a given mesh entity. We note that the mesh entities, defined by class `MsqMeshEntity`, are the only exception in our architecture. Mesh entities, i.e. triangles, tetrahedrons, quadrangles and hexahedrons, are implemented in a single class, without the use of dynamic polymorphism, to eliminate the performance impact of runtime resolution.

In this section, we describe the details of the core classes `QualityMetric`, `ObjectiveFunction`, `QualityImprover`, `TerminationCriterion`, `QualityAssessor`, `MeshSet`, `PatchData`, and `InstructionQueue`. In each case, we give details of the design and discuss the functionality currently implemented in Mesquite. Often, we will refer to software patterns widely used in object-oriented design — the details of those design patterns are described in [21].

3.1 Quality Metrics

In Mesquite, the `QualityMetric` class provides a measure of the quality of individual mesh entities. Quality metrics can evaluate either element quality (for example, the mean ratio shape quality metric) or vertex quality (for example, the sum of the adjacent edge lengths squared can be used as a measure of vertex smoothness). The primary functionality associated with the `QualityMetric` class is the 'evaluate' function which returns a single quality value for a given mesh entity.

To increase the flexibility of the `QualityMetric` class, we also provide a number of mechanisms that allow the user to modify the metric in a variety of ways. For example, the condition number quality metric for a quadrilateral or hexahedral element is computed by evaluating the condition number at a number of sample points in the element. Mesquite allows the user to select which set of sample points are used in this calculation (*e.g.*, the element's vertices) and how these values are combined to form a single metric value (*e.g.*, linear averaging). In addition, Mesquite is currently being extended to allow the user to reference certain quality metrics to a non-isotropic element.

In addition to the quality metric function value, the `QualityMetric` class also provides the gradient and Hessian information needed for many optimization algorithms. Numerical approximations of the gradient and Hessian are automatically provided by the `QualityMetric` base class. Concrete instantiations can also optionally include analytical expressions which are potentially more computationally efficient. For example, in the case of the mean ratio quality metric implementation, the analytic calculation is approximately twice as fast as the numerical computation although this improved efficiency often comes at a higher implementation cost. The cost of implementing the analytical gradients and Hessians can often be alleviated, however, by the use of automatic differentiation tools (for example, [22]).

Mesquite also allows the user to scale metric values or combine multiple metrics together to form a composite metric. However, only metrics which are evaluated on the same type of mesh entity can be composed together. That is, Mesquite does not allow element-based metrics and vertex-based metrics to be added or multiplied together because the result is not a meaningful measure of either element or vertex quality.

Currently Available Quality Metrics. There are currently nine quality metrics available within Mesquite, and these are listed in Table 3.1 (detailed definitions of the metrics are given in [23]). These quality metrics are grouped by the type of mesh properties that they measure, in particular: shape, smoothness, volume, and untangle. There are also two composite metrics which allow users to multiply two metrics' values together or to raise a single metric's value to a given power. The latter allows for negative powers and can therefore be used to obtain the inverse of any Mesquite quality metric. We note that the implementation of the mean ratio metric has been extensively optimized, and analytical gradients and Hessians are available for that function. Other metrics currently use numerical gradients and Hessians. Future Mesquite development will include the implementation of metrics falling under other group headings such as orthogonality, shear, and alignment.

3.2 Objective Functions

While the `QualityMetric` class provides a way to evaluate the properties of individual mesh entities, the `ObjectiveFunction` class provides a way of combining those values into a single number for the domain of the optimization problem. This domain can either be the entire mesh or a sub-mesh containing a subset of the free vertices. For example, one available objective function template f is the ℓ_2^2 function which is the standard ℓ_2 vector norm squared. Given an element-based quality metric q and a mesh sub-domain E, the mesh quality objective function $\mathcal{F}$ would be the composition of the template objective function and the quality metric function

$$\mathcal{F}(\mathbf{x}) = f \circ q(\mathbf{x}) = \sum_{i \in E} (q_i(\mathbf{x}))^2 \quad . \tag{1}$$

The `ObjectiveFunction` derived class computes the value of $\mathcal{F}$ and, for f and q satisfying the appropriate smoothness conditions, the mesh quality objective function's gradient and Hessian with respect to the vertex positions. As with `QualityMetric`, Mesquite allows the gradient of $\mathcal{F}$ to be calculated either analytically or numerically. Computing the gradient of $\mathcal{F}$ numerically is computationally expensive but requires only the quality metric values. If the gradient is calculated analytically, the first derivative of the template objective function f' and the quality metric gradient ∇q are both required.[1] To obtain the gradient of $\mathcal{F}$, the chain rule is applied

$$\nabla \mathcal{F}(\mathbf{x}) = \amalg_{i \in E} \left[\nabla q_i(\mathbf{x}) (f' \circ q(\mathbf{x})) \right] \quad , \tag{2}$$

where $f' \circ q(\mathbf{x})$ is a scalar and $\amalg_{i \in E}$ denotes the assembly over all the elements or vertices, for element-based

[1]The quality metric gradients ∇q can be provided either numerically or analytically.

Metric	Group	Mesh Type	Feasibility Region	Entity Type
Area Smoothness	Smoothness	Any	No	Elements
Aspect Ratio	Shape	Tri/Tet	No	Elements
Composite Mult.	Composite	Any	Yes/No	Either
Composite Power	Composite	Any	Yes/No	Either
Cond. Num.	Shape	Any	Yes	Elements
Corner Jacobian	Volume	Any	No	Elements
Edge Length	Smoothness	Any	No	Vertices
Edge Length Range	Smoothness	Any	No	Vertices
Mean Ratio	Shape	Any	Yes	Elements
Untangle Beta	Untangle	Any	Yes	Elements
Vert. Cond. Num.	Shape	Any	Yes	Vertices

Table 1: List of the current Mesquite quality metrics. The table also indicates the metric group, the mesh types for which the metric is valid, whether the metric is only valid within a feasible region, and the type of entity for which the metric is defined (elements or vertices). Note that there may be a feasible region for composite metrics depending on whether the underlying metrics require such a constraint.

or vertex-based quality metrics respectively. Analytical gradients have been implemented for all of the continuously differentiable template objective functions in Mesquite, and an analytical Hessian calculation has been implemented for the ℓ_p^p template objective function.

Available Template Objective Functions. Mesquite currently has objective function templates for the standard ℓ_P (where P is a positive integer) and ℓ_∞ vector norms. A separate template is provided for ℓ_P^P, a function which has several nice properties including a sparse Hessian matrix. All of Mesquite's quality improvers are designed to minimize a given objective function. For objective functions that need to be maximized (*e.g.* an ℓ_1 objective function using an inverted mean ratio metric), the function value is multiplied by negative one to obtain the equivalent minimization problem.

Mesquite has four composite objective functions that allow the user to add and multiply objective functions by each other or by scalar values. These can be used to modify the optimization problem, to ease interpretation of the objective function value, or to ensure compatibility with termination criterion based on objective function values. Unlike quality metrics, two objective functions can be combined even if the underlying quality metrics are defined on different entity types. That is, an objective function which operates on a vertex-based quality metric can be added to an objective function which operates on an element-based quality metric. This allows the user to have the maximum flexibility in defining an objective function with which to measure the quality of a mesh.

3.3 Quality Improvers

The mesh quality improvement algorithms are a crucial component of the Mesquite framework. The two main types of improvement schemes designed into Mesquite are `VertexMover` and `TopologyModifier` for vertex relocation or topology modification, respectively. These methods take as input an `ObjectiveFunction` and often make extensive use of the gradient and Hessian information provided therein. When writing a new algorithm, the concrete `QualityImprover` always acts on an `ObjectiveFunction` pointer to retrieve the function value and gradient for a certain mesh and concrete `ObjectiveFunction / QualityMetric` combination.

A new quality improver is defined by inheriting from either the `VertexMover` or the `TopologyModifier` abstract class. Both classes are intermediate abstract classes inheriting from the `QualityImprover` class (see Figure 1). One important aspect of both vertex movers and topology modifiers is the ability to seamlessly perform their operations globally on the entire mesh or on local sub-patches of the mesh. The behavior is chosen by the user at compile time using the `set_patch_type` function from `QualityImprover`. The primary functionality in these intermediate classes is the implementation of the `loop_over_mesh` virtual function which shields the optimization algorithm developer from the need to distinguish between local or global patches. The appropriate mesh information is gathered into a "patch" by the `MeshSet` and `PatchData` classes and given to the `QualityImprover` in the `loop_over_mesh` function. The `VertexMover` base class also checks the outer termination criterion to stop iterating over mesh subsets (see section 3.4) and updates the application mesh after optimizing a patch.

Available Quality Improvement Algorithms. There are currently three major concrete optimization algorithms implemented as `VertexMovers` in Mesquite: the conjugate gradient algorithm, the feasible Newton algorithm, and the active set algorithm. The optimization of mesh topology has been accounted for in the architecture, but no implementation is available yet.

Conjugate Gradient Algorithm. This algorithm is appropriate for optimizing any combination of C^1 objective functions and quality metrics (see [5] for more details). The conjugate gradient algorithm has a linear convergence property for most problems. By using the Polak-Ribière scheme to select a search direction which is a combination of the gradient at the current iteration with the gradient from one or more previous iterations, it avoids the zigzagging behavior exhibited by the steepest descent algorithm when the equal cost surfaces of the objective function are elongated (e.g. in narrow valleys). The algorithm requires the objective function value and gradient and can be used on mesh patches of any size.

Feasible Newton Algorithm. Newton's method minimizes a quadratic approximation of a non-linear objective function. Newton's method is known to converge super-linearly near a non-singular local minimum. Mesh-optimization problems that are performed within the neighborhood of the minimum are a perfect application for Newton's method. The algorithm requires the objective function value, gradient, *and* Hessian information. In Mesquite this algorithm can be used with mesh patches of any size, making it appropriate to optimize all vertex positions simultaneously for any C^2 objective function with a sparse Hessian[2]. In practice, users will often observe an order of magnitude improvement in computation time when using the feasible Newton algorithm instead of the conjugate gradient algorithm, making feasible Newton a worthwhile choice when applicable (see [5] for more details).

Active Set Algorithm. The active set algorithm has been developed for non-continuously differentiable objective functions such as those computing the maximum value of a quality metric within a patch of elements. This method is based on a non-smooth steepest descent algorithm which efficiently computes a search direction and step size based on the gradients of the values contained in the active set. Currently, this algorithm works on a patch of triangular or tetrahedral elements that contain a single free vertex. Repeated sweeps over the free vertices in the mesh leads to overall mesh improvement (see [24] for more details).

[2]Note that objective functions that lead to non-sparse Hessians entail a prohibitive memory cost in Mesquite; ℓ_p for $p \geq 2$ is such a function. The number of variables in the objective function is the number of degrees of freedom of the mesh, which must be squared to obtain the number of entries in a dense Hessian.

3.4 Termination Criterion

Mesquite's `TerminationCriterion` class contains functionality to customize the termination of the mesh quality improvement process. As mentioned previously, many quality improvement algorithms can perform mesh optimization on either the global mesh or on sub-meshes. In the latter case, the algorithm must be capable of determining when to terminate two processes: 1) an inner criterion is used to terminate the optimization algorithm on a sub-mesh and 2) an outer criterion is used to terminate the iteration over the sub-meshes. Typically, quality improvers that support both local and global optimizations always use two termination criterion. In the global case, the outer criteria is set to terminate the optimization process after one iteration.

Available Termination Criterion. A wide range of cost, quality, and progress-centric termination criterion types have been studied. These criteria have two basic types - absolute or relative - depending on whether or not the criterion is scale dependent. Fourteen of these have been implemented in `TerminationCriterion`. Among the implemented types are criteria which terminate optimization procedures due to exceeding a set number of iterations, exceeding an allotted amount of time, or reaching a mesh with a sufficiently small objective function gradient. Any combination of the available criteria can be set on a given `TerminationCriterion` object. Compound criteria types consist of statements joined by 'OR', for which the optimization process will be terminated when *any* of the criteria have been satisfied. Currently we have found little use for compound criteria joined by 'AND', but this also could be implemented.

3.5 Quality Assessors

Mesquite's `QualityAssessor` class encapsulates functionality to evaluate quality metric values for a given mesh, to accumulate statistical information about those values, and to report that data to the user. In particular, a `QualityAssessor` object takes a `QualityMetric` class as input to evaluate a given mesh and then reports information like the maximum, average, and standard deviation of those values.

3.6 Mesh Data Classes

There are two mesh data classes in Mesquite, `MeshSet` and `PatchData`, which have been designed to meet two different needs. The `MeshSet` class is a container that holds pointers, or handles, to the meshes provided by the application. It does not store any detailed information about the mesh such as vertex coordinates or element connectivities, but provides the mechanisms necessary to obtain this information from the application through a well-defined, flexible API (see Section 3.6.1). Detailed mesh information obtained through the MeshSet API is stored only in the `PatchData` class for the sub-patches given to the quality improver procedures. The `PatchData` class makes Mesquite scalable in that prohibitive memory costs associated with making a copy of a large application mesh can be avoided by dividing the mesh into patches on which to perform the optimization sequentially.

3.6.1 MeshSet Interactions with Application Meshes

The `MeshSet` class is responsible for gathering information from the application's mesh and geometry and placing this information into `PatchData`. Because Mesquite is designed as a library to work on a broad assortment of mesh and element types on complex geometrical domains, a general, data-structure neutral API is needed. In general, Mesquite requires access to basic information about the mesh such as the number of vertices and elements in the mesh, the vertex locations, and the element connectivities. To move the vertex locations, Mesquite needs to set the vertex coordinate positions, and eventually, to perform swapping operations Mesquite will need to add and delete various mesh entities. In addition, for smoothing meshes on complex surfaces, access to operations on the underlying solid model such as normal information and closest point information are required to ensure vertices are constrained to the surface.

To allow an application to expose this information to Mesquite, we have defined a set of interfaces (C++ abstract base classes) that are specifically designed for mesh quality improvement needs. There are four such interfaces: Mesh, VertexIterator, ElementIterator, and MeshDomain.

- *Mesh:* The Mesh interface represents the set of mesh entities that are to be operated on. It is through this interface that one retrieves information about the mesh and its entities. Examples of functionality provided by this interface include: retrieving the number of elements in the mesh, determining which elements contain a particular vertex, and modifying vertex coordinates.

- *VertexIterator:* The VertexIterator provides access to each vertex in a mesh. A VertexIterator is obtained from a Mesh object, and is used to iterate through the list of all vertices in the Mesh from which it was obtained.

- *ElementIterator:* The ElementIterator provides access to each element in a mesh. Other than the type of entity it exposes, it is identical to the VertexIterator.

- *MeshDomain* The MeshDomain represents the set of geometric domains to which the mesh may be constrained. The MeshDomain interface enables an application to restrict the locations to which a vertex can be moved, such as constraining a vertex to a surface. Through the MeshDomain interface, Mesquite's algorithms can also obtain a domain's normal vector, which aides validity checking and decision making during the quality improvement process.

These interfaces are data-structure neutral and use only primitive data types; an application may implement the Mesquite interfaces without changing its existing mesh data structures. Instead of representing mesh entities with complex data structures or with typed pointers, entities are identified with opaque values called handles. Each mesh entity has a unique handle value, but otherwise handles have no intrinsic meaning to Mesquite.

Mesquite can also use the mesh interfaces currently being developed through the TSTT center. This interface definition effort focuses on providing access to information pertaining to low level mesh objects such as vertices, edges, faces, and regions through both array-based and iterator-based mechanisms. It is designed to support existing packages such as CUBIT, NWGrid, PAOMD, and Overture. Considerations such as data neutrality, language interoperability (achieved through use of the SIDL/Babel tools from LLNL [25]), and achieving consensus within a large group of participants is paramount (see [17], [18], [26], and [19]). This interface definition effort is evolving, and the Mesquite team is actively participating to ensure that our needs for mesh quality improvement are adequately and efficiently addressed. A TSTT-based implementation of the Mesquite interfaces will be available soon. As such, any tool that exposes its mesh through the TSTT interfaces can be used with Mesquite without additional development.

The Mesquite-specific interfaces are fully compatible with the current TSTT mesh and geometry interfaces, and in fact, Mesquite's approach to data structure neutrality is directly derived from the TSTT interfaces. Although similar in spirit to the TSTT interface, the Mesquite-specific interface is not as general, and therefore consist of fewer functions and does not require additional tools such as Babel.

3.6.2 PatchData Interactions with QualityImprovers

Quality improvers are written to relocate nodes or modify topology within a `PatchData`, without any need to know whether the `PatchData` corresponds to the whole mesh or a subset of it. The `PatchData` information is generated by the `MeshSet` class with the `get_next_patch` function — the equivalent of an iterator over a series of patches covering the mesh. The user can set Mesquite to use different types of `PatchData`, ranging from a patch of elements containing one particular vertex to a patch of vertices connected to a central vertex through edges or a unique patch that covers the whole MeshSet.

`PatchData` and its associated classes (e.g. `PatchDataVerticesMemento`) provide much functionality to the optimization algorithms. Memento patterns [21] can remember the state of a `PatchData` geometry or topology at a given iteration and restore the `PatchData` to that state later. Simple functions can move the `PatchData` n vertices in a direction $d \in \mathbb{R}^{3n}$ while constraining the boundary vertices to their geometrical surface.

3.7 The Instruction Queue

The `InstructionQueue` class allows a sequence of operations such as quality assessment and quality improvement to be performed on a `MeshSet` object. The `InstructionQueue` provides a convenient framework to shield the user from the algorithm syntax and to ensure a consistent use of the Mesquite capabilities. One or more quality improvers can be associated with an `InstructionQueue`, but one must be designated as the *master* quality improver that determines the ultimate improvement goal. All progress made by Mesquite will be measured against the quality metrics set in the master quality improver. To improve the effectiveness and efficiency of the mesh quality improvement process, several quality improvers can be used as 'pre-conditioners' for the master quality improver. For example, a user may precede an optimization-based master quality improver with a mesh untangler and/or Laplacian smoothing.

Some predefined `InstructionQueue` objects, called *wrappers*, are available for high-level or novice users. Those typically consist of a quality assessor, followed by a mesh pre-conditioner such as an untangler, followed by a master quality improver, and finally another quality assessor. Once an InstructionQueue has been defined, a single call to `run_instructions` will perform all the contained operations. We note that once an `InstructionQueue` has been defined, it can be used for several `MeshSet` objects.

3.8 Code Testing Framework

Another integral part of the Mesquite framework is the code testing infrastructure. Several testing methodologies have been included within Mesquite. Unit testing is extensively used to facilitate development and ensure low-level robustness. Functional testing is used to ensure that user case scenarios run smoothly.

We use a broad definition for unit tests in Mesquite. Any test performed on a class without need for the entire Mesquite framework is considered a unit test. This encompasses simple assertions like checking the result of the multiplication of a matrix $A \in \mathbb{R}^{3\times 3}$ by a vector $v \in \mathbb{R}^3$ to more complex assertions such as checking that a concrete `QualityImprover` correctly repositions a free vertex in a simple patch for a given objective function and quality metric. Mesquite uses a readily available testing framework called CppUnit [27] — essentially the well known jUnit testing framework ported to C++.

Applications using Mesquite may also find the testing framework useful when verifying their additions to the code. In particular, applications that prefer to implement Mesquite's mesh interface instead of using the TSTT mesh interface can heck their implementation with the corresponding unit test collection available in Mesquite. In addition, analytic gradient and Hessian implementations can be checked against the numerical version provided by the `QualityMetric` base class using readily available unit tests.

In addition to unit tests, the Mesquite test suite also includes a range of functional tests. While these tests also use the CppUnit framework, they differ from unit tests in that functional tests require the entire Mesquite framework. The functional tests are complete and often complex mesh optimization problems. These tests are intended to ensure that the individual units of the Mesquite code work together correctly. Performing these tests not only helps validate the code but also allows developers to evaluate the effects of code modifications in terms of Mesquite's accuracy and efficiency on 'real world' problems.

4. USER PROGRAMMING INTERFACES

We provide a number of mechanisms that lower the "expertise" barrier for using Mesquite. In the simplest usage scenario, a TSTT mesh pointer is passed to Mesquite and improved using a default set of metrics and strategies determined by the mesh type (Section 4.2). If desired, the user can guide the mesh improvement process by setting a few parameter values indicating preferred metrics and strategies from a list of provided functionalities. Once these parameters are defined, Mesquite optimizes the mesh without further input from the user. Additional interfaces are provided for those who desire advanced functionalities, user-defined metrics or objective functions, or more control over the optimization process (Section 4.1).

In sections 4.1 and 4.2, we assume that a `MeshSet` has already been instantiated and a TSTT mesh handle given to it to populate the database as shown below:

```
  // Create a TSTT mesh
TSTT::Mesh wing = TSTT::Mesh::_create();
wing.load(``wing_file.xx'');

  // Create a Mesquite MeshSet
Mesquite::MeshSet mesh_set;
mesh_set.add_mesh(wing);
```

4.1 Low-level API

Extensive control over the mesh quality improvement process is provided through the direct use of the Mesquite classes. In particular, the user can specify the quality metric, objective function template, and optimization algorithm by instantiating particular instances of each. For each, various options such as numerical or analytical gradient and Hessian evaluations or the patch size can be selected. Furthermore, the user can fine tune the optimization algorithm performance by creating and setting the parameters of the termination criterion for both inner and outer iterations.

Once these core objects have been created and customized, the user creates an instruction queue and adds one or more quality improvers and quality assessors. The mesh optimization process is initiated with the `run_instructions` method on the instruction queue class. For example, in the code given below, a mean ratio quality metric is created and analytic gradient and hessian computations are selected. The mean ratio quality metric is provided as input to the objective function template constructor, which will also use analytical gradient computations. The feasible Newton vertex mover is created and the patch size is set to be the global mesh. A termination criterion that checks the value of the ℓ_2 norm of the gradient of the objective function is added to the quality improver. Finally an instruction queue that checks the quality of the mesh both before and after the feasible Newton algorithm runs is created and run on the `mesh_set`.

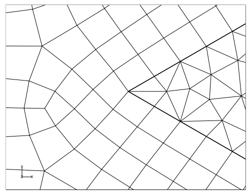
```
  // creates a mean ratio quality metric ...
ShapeQM* m_ratio = MeanRatioQM::create_new();
m_ratio->set_gradient_type(ANALYTIC_GRADIENT);
m_ratio->set_hessian_type(ANALYTIC_HESSIAN);

  // sets the objective function template
LPtoPTemplate obj_func(m_ratio, 2);
obj_func->set_gradient_type(ANALYTIC_GRADIENT);

  // creates the optimization procedures
FeasibleNewton f_newton(&obj_func);

  //performs optimization globally
f_newton->set_patch_type(GLOBAL_PATCH);

  // creates a termination criterion and
  // add it to the optimization procedure
  // outer loop: default behavior: 1 iteration
  // inner loop: stop if gradient norm < eps
TerminationCriterion tc_inner;
tc_inner.add_criterion_type(GRAD_L2_NORM, e-4);
f_newton->set_inner_terminate_crit(&tc_inner);

  // creates a quality assessor
QualityAssessor m_ratio_qa(&m_ratio,AVERAGE);

  // creates an instruction queue
InstructionQueue queue1;
queue1.add_quality_assessor(&m_ratio_qa);
queue1.set_master_quality_improver(&f_newton);
queue1.add_quality_assessor(&m_ratio_qa);

  // launches optimization on the mesh_set
queue1.run_instructions(mesh_set);
```

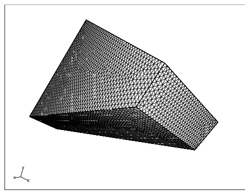
4.2 High-level API — Wrappers

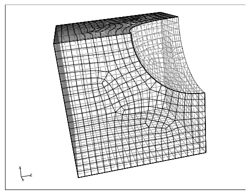
To improve meshes with a minimum number of Mesquite function calls, we have provided a set of wrapper classes that encapsulate the most commonly used combination of quality metrics, improvement algorithms and stopping criterion. The wrappers inherit from the `InstructionQueue` class and set the algorithms used in their constructors. Using these wrappers, only two lines of code are required to improve a `MeshSet`.

For example, the ShapeImprovementWrapper shown below performs operations similar to the low-level code given in section 4.1, but without providing access to low-level features such as setting a specific termination criterion. This wrapper first uses the mean ratio quality metric to assess and report the quality of the mesh elements. It then untangles the mesh if necessary and improves the mesh with the feasible Newton algorithm applied to the composition of the mean ratio quality metric with the ℓ_2^2 objective function. Finally, the mesh quality is assessed and reported again.

```
  // Creates wrapper and improves mesh
Mesquite::ShapeImprovementWrapper shape_improver;
shape_improver.run_instructions(mesh_set);
```


(a)


(b)


(c)

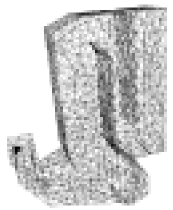

(d)

Figure 2: (a) A 'zoomed-in' picture of a hybrid triangular and quadrilateral mesh. (b) A tetrahedral mesh. (c) A hybrid tetrahedral and hexahedral mesh. (d) A tetrahedral mesh of a tire incinerator. Meshes (a), (b), and (c) were generated with CUBIT (see [17]).

5. EXAMPLES

Although currently in a pre-release state, Mesquite has been used to improve a variety of mesh types. As mentioned previously, Mesquite is designed to efficiently handle a mesh of any size and element type. While not all element types are currently implemented, the code can handle triangular, quadrilateral, tetrahedral, and hexahedral elements, as well as hybrid combinations of those types. Examples of optimizations on each of these mesh types are given in Table 2. The selected meshes also represent a wide range of initial mesh qualities with two of the initial meshes containing invalid elements. Each mesh was optimized using the same objective functions and algorithms. They were first untangled (a process which has no effect on the initially valid meshes), and they were then optimized with respect to the mean ratio metric using the conjugate gradient algorithm and an ℓ_2 objective function template. The table shows the average mean ratio metric value of both the initial and the optimized meshes.

Mesh	Vertices	Init. μ_{avg}	Final μ_{avg}
Tri.	10300	1.036781	1.033657
Quad.	267	∞	2.485082
Hybrid 2D	263	1.166410	1.103776
Tet.	21156	1.168008	1.119870
Hex.	12753	3.497623	3.096999
Hybrid 3D	9200	∞	1.098384

Table 2: Meshes with different element types optimized using Mesquite. The average mean ratio, μ_{avg}, is given for the initial and the final mesh. The hybrid 2-dimensional mesh, the tetrahedral mesh, and the hybrid 3-dimensional mesh are shown in Figures 2 (a), (b), and (c), respectively. The other meshes are not shown.

Figures 2 (b) and (d) show two tetrahedral meshes. The first mesh, given in Figure 2 (b) and Table 2, is a wedge-shaped block containing 112,393 tetrahedral elements and 21,156 vertices. The second mesh, given in Figure 2 (b), contains 11,098 elements and 2,570 vertices and is the mesh of a tire incinerator. These two meshes were optimized using three different quality improvers and objective function templates. Each optimization was based on the mean ratio quality metric. The quality of the resulting meshes is given in Table 3.

Mesh	Algorithm	Func.	max. μ	avg. μ
Tire	Initial Mesh	-	22.6413	1.30354
	Active Set	ℓ_∞	6.11002	1.37703
	Conj. Grad.	ℓ_1	6.11002	1.25832
	Feas. Newton	ℓ_2^2	6.11002	1.25913
Wedge	Initial Mesh	-	4.19031	1.16800
	Active Set	ℓ_∞	1.54073	1.18224
	Conj. Grad.	ℓ_1	2.26029	1.11937
	Feas. Newton	ℓ_2^2	2.04328	1.11950

Table 3: Metric values for tetrahedral meshes of a tire incinerator (Tire) and a wedge-shaped block (Wedge) optimized using three different quality improvement algorithms and objective functions. Each objective function uses the element-based mean ratio metric, denoted as μ. Tire is shown in Figure 2 (d), and Wedge is shown in Figure 2 (b).

Mesquite has also been used to optimize a geodesic mesh provide by the climate group at Colorado State University. This mesh is a dense, triangular element mesh called a twisted icosahedron grid. It is formed by refining the triangles of an icosahedron and projecting the new vertices to the unit sphere. This meshing scheme produces a relatively regular mesh with slight variations in the elements' shapes and sizes. Mesquite was therefore used to decrease the magnitude of those variations to produce a smoother mesh in terms of both element areas and edge lengths.

6. SUMMARY AND FUTURE WORK

The Mesquite design evolved to address the high level goals of flexibility, comprehensiveness, efficiency, and interoperability through a careful balancing of upper-level C++ classes and a low-level optimizable kernel. Years of prior experience with a variety of mesh quality improvement problems guided the design so that, although the design is still evolving (as is the mathematical framework which supports it), we expect the addition of new capabilities to impact the design relatively little. Mesquite has progressed rapidly during the past year to include a considerable number of mesh quality metrics, objective function templates, solvers, and termination criteria. Several important development objectives remain such as the inclusion of concrete 'topology modifiers' and new quality metrics to guide the evolution of deforming meshes, anisotropic smoothing, and R-type adaptivity based on solution features or error indicators. Preliminary applications of Mesquite include the smoothing of geodesic meshes and the improvement of mesh quality for increasing Tau3P abort time. Near-term code development is driven primarily by SciDAC applications, however, we anticipate many other eventual uses.

Acknowledgments The authors wish to thank Todd Munson at Argonne National Laboratory for his contributions on the feasible Newton solver and Todd Ringler at Colorado State University for the geodesic meshes.

References

[1] Hansbo P. "Generalized Laplacian Smoothing of Unstructured Grids." *Comm. Num. Meth. Engr.*, vol. 11, 455–464, 1995

[2] Winslow A. "Numerical Solution of the quasilinear Poisson equations in a nonuniform triangle mesh." *J. Comp. Phys.*, vol. 2, 149–172, 1967

[3] Knupp P. "Achieving Finite Element Mesh Quality via Optimization of the Jacobian Matrix Norm and Associated Quantities. Part I - A Framework for Surface Mesh Optimization." *Int'l. J. Numer. Meth. Engr.*, vol. 48, no. 3, 401–420, 2000

[4] Freitag L., Knupp P. "Tetrahedral mesh improvement via optimization of the element condition number." *Intl. J. Numer. Meth. Engr.*, vol. 53, 1377–1391, 2002

[5] Freitag L., Knupp P., Munson T., Shontz S. "A Comparison of optimization software for mesh shape-quality improvement problems." *Proceedings of the 11th International Meshing Roundtable*, pp. 29–40. Sandia National Laboratories, Ithaca, NY, 2002

[6] Canann S., Stephenson M., Blacker T. "Optismoothing: An optimization-driven approach to mesh smoothing." *Finite Elements in Analysis and Design*, vol. 13, 185–190, 1993

[7] Canann S., Tristano J., Staten M. "An approach to combined Laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes." *Proceedings of the 7th International Meshing Roundtable*, pp. 479–494, 1998

[8] Parthasarathy V.N., Kodiyalam S. "A constrained optimization approach to finite element mesh smoothing." *Finite Elements in Analysis and Design*, vol. 9, 309–320, 1991

[9] Amezua E., Hormaza M.V., Hernandez A., Ajuria M.B.G. "A method of the improvement of 3D solid finite-element meshes." *Advances in Engineering Software*, vol. 22, 45–53, 1995

[10] Joe B. "Three-dimensional triangulations from local transformations." *SIAM Journal on Scientific Computing*, vol. 10, 718–741, 1989

[11] Joe B. "Construction of three-dimensional improved quality triangulations using local transformations." *SIAM Journal on Scientific Computing*, vol. 16, 1292–1307, 1995

[12] Edelsbrunner H., Shah N. "Incremental topological flipping works for regular triangulations." *Proceedings of the 8th ACM Symposium on Computational Geometry*, pp. 43–52. 1992

[13] Freitag L. "Users Manual for Opt-MS: Local Methods for Simplicial Mesh Smoothing and Untangling." Tech. Rep. ANL/MCS-TM-239, Argonne National Laboratory, Chicago, IL, 1999

[14] Brown D., Freitag L., Glimm J. "Creating Interoperable Meshing and Discretization Software: The Terascale Simulation Tools and Technologies Center." *Proceedings of the Eighth International Conference on Grid Generation Methods for Numerical Field Simulations.* Honolulu, HI, 2002

[15] Randall D., Ringler T., Heikes R., Jones P., Baumgardner J. "Climate Modeling with Spherical Geodesic Grids." *Computing in Science and Engineering*, vol. 4, no. 5, 32–41, 2002

[16] Folwell N., Knupp P., Brewer M. "Increasing Tau3P Abort-Time via Mesh Quality Improvement." *Proceedings of the 12th International Meshing Roundtable*, 2003

[17] "CUBIT Mesh Generation Toolsuite." URL: http://cubit.sandia.gov, 2003. Sandia National Laboratories

[18] Brown D.L., Henshaw W.D., Quinlan D.J. "Overture: An object oriented framework for solving partial differential equations." *Scientific Computing in Object-Oriented Parallel Environments*, vol. 1343. Springer Lecture Notes in Computer Science, 1997

[19] "The P3D Code Development Project." URL: http://www.emsl.pnl.gov:2080/nwgrid/, 2003. Pacific Northwest National Laboratory

[20] Freitag L., Knupp P., Leurent T., Melander D. "MESQUITE Design: Issues in the Development of a Mesh Quality Improvement Toolkit." *Proceedings of the Eighth International Conference on Grid Generation Methods for Numerical Field Simulations*, pp. 159–168. Honolulu, HI, 2002

[21] Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns, Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995

[22] Bischof C.H., Roh L., Mauer-Oats A.J. "ADIC: an extensible automatic differentiation tool for ANSI-C." *Software Practice and Experience*, vol. 27, no. 12, 1427–1456, 1997

[23] Knupp P. "Algebraic Mesh Quality Metrics." *SIAM J. Sci. Comput.*, vol. 23, no. 1, 193–218, 2001

[24] Freitag L., Plassmann P. "Local Optimization-based Simplicial Mesh Untangling and Improvement." *Intl. J. Num. Methods in Engr.*, vol. 49, 109–125, 2000

[25] Kohn S., Kumfert G., Painter J., Ribbens C. "Divorcing Language Dependencies from a Scientific Software Library." Tech. Rep. UCRL-JC-140349, Lawrence Livermore National Laboratory, Livermore, CA, 2001

[26] Remacle J.F., Klass O., Flaherty J.E., Shephard M.S. "Parallel Algorithm oriented mesh database." *Proceedings of the 10th International Meshing Roundtable*, pp. 197–206. Sandia National Laboratories, Newport Beach, CA, 2001

[27] Sommerlade E., Feathers M., Lacoste J., Hoffmann J., Lepilleur B., Bakker B., Robbins S. "CppUnit – The C++ Unit Testing Library." `http://cppunit.sourceforge.net`