ANL/MCS-TM-239

# Users Manual for Opt-MS: Local Methods for Simplicial Mesh Smoothing and Untangling

by

*Lori Freitag*

MATHEMATICS AND COMPUTER SCIENCE DIVISION

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL/MCS-TM-239

# Users Manual for Opt-MS: Local Methods for Simplicial Mesh Smoothing and Untangling

by

*Lori Freitag*

Mathematics and Computer Science Division

Technical Memorandum No. 239

April 1999

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

## Contents

# Users Manual for Opt-MS: Local Methods for Simplicial Mesh Smoothing and Untangling

by

*Lori Freitag*

## Abstract

Creating meshes containing good-quality elements is a challenging, yet critical, problem facing computational scientists today. Several researchers have shown that the size of the mesh, the shape of the elements within that mesh, and their relationship to the physical application of interest can profoundly affect the efficiency and accuracy of many numerical approximation techniques. If the application contains anisotropic physics, the mesh can be improved by considering both local characteristics of the approximate application solution and the geometry of the computational domain. If the application is isotropic, regularly shaped elements in the mesh reduce the discretization error, and the mesh can be improved *a priori* by considering geometric criteria only.

The Opt-MS package provides several local node point smoothing techniques that improve elements in the mesh by adjusting grid point location using geometric criteria. The package is easy to use; only three subroutine calls are required for the user to begin using the software. The package is also flexible; the user may change the technique, function, or dimension of the problem at any time during the mesh smoothing process. Opt-MS is designed to interface with C and C++ codes, and examples for both two- and three-dimensional meshes are provided.

## 1 Introduction

Automatic mesh generation tools are often used to decompose complex geometries into a union of simple geometric shapes, including triangles and quadrilaterals in two dimensions and tetrahedron and hexahedron in three dimensions. Unfortunately, meshes generated in this way can contain poorly shaped or distorted elements that result in numerical difficulties during the solution process [11]. If the application contains anisotropic physics, the mesh can be improved by considering both local characteristics of the approximate application solution and the geometry of the computational domain. If the application is isotropic, regularly shaped elements in the mesh reduce the discretization error, and the mesh can be improved *a priori* by considering geometric criteria only. Several techniques have been developed to perform this improvement, including local reconnection techniques such as edge or face swapping [6], [15], [17], node point smoothing [2], [4], [20], and combinations of the two.

The Opt-MS software package provides several local smoothing techniques that adjust the position of one vertex at a time to obtain improvement in a neighborhood around that vertex. In particular, the Opt-MS routines work on a local submesh consisting of a free vertex, $v$, its incident elements, $t_i$, and adjacent vertices, $v_i$, as shown in Figure 1. Mesh smoothing algorithms can be heuristic or formulated as a rigorous optimization problem, but the goal of all algorithms is to move the free vertex to a new position that improves the quality of the local submesh by some measure. For example, if the minimum angle in the mesh provides a measure of quality, then the local submesh on the right has higher quality than the submesh on the left because the minimum angle is closer to 60° than to 20°. Local methods achieve this improvement by adjusting only the position of the free vertex, $v$, in each local submesh; adjacent vertex locations remain unchanged. Some number of sweeps over the adjustable vertices are performed to achieve overall improvement in the mesh. The local techniques included in the Opt-MS package are Laplacian smoothing [8, 19], an optimization-based approach formulated and implemented at Argonne National Laboratory, and various combinations of the two. More detailed information regarding the smoothing approaches, quality metrics, and typical results are provided in Section 2.

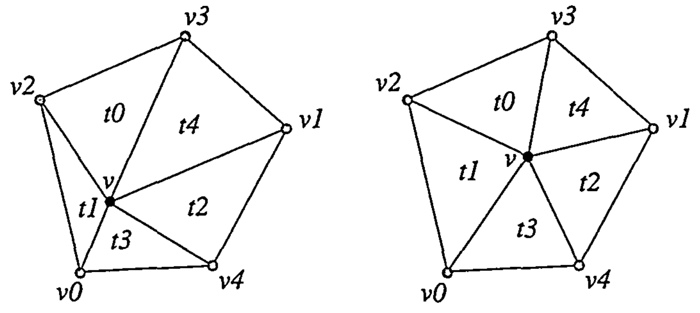

Figure 1: A local submesh consisting of a free vertex, $v$, to be moved and its incident elements, $t0 \dots t4$. The figure on the right shows the same local submesh after node point smoothing.

In addition to mesh smoothing, the Opt-MS package includes a technique for untangling simplicial meshes with valid connectivity, but invalid or inverted elements (those with negative area). An invalid local submesh is shown in Figure 2. The submesh on the left consists of a free vertex, $v$ and its seven adjacent vertices and triangles, all of which are valid because $v$ lies inside the *feasible region* (shaded gray). The submesh on the right consists of the same adjacent vertices, but the free vertex lies outside the feasible region, causing triangles $t1$ and $t2$ to have negative area. The untangling method provided in Opt-MS is formulated as a linear programming problem on local submeshes. As with local smoothing techniques, some number of sweeps through the local submeshes are performed until the mesh is untangled or a maximum number of iterations has been exceeded. A brief description of the formulation, solution technique, and typical results are given in Section 3.

The Opt-MS library has been designed to improve both two- and three-dimensional simplicial meshes through a small number of API routines. The primary functionality of the Opt-MS package and examples of its use, including the required user input, skeleton example codes, and typical output are described in Section 4. The configuration and compilation of the library is described in detail in Section 5. The API

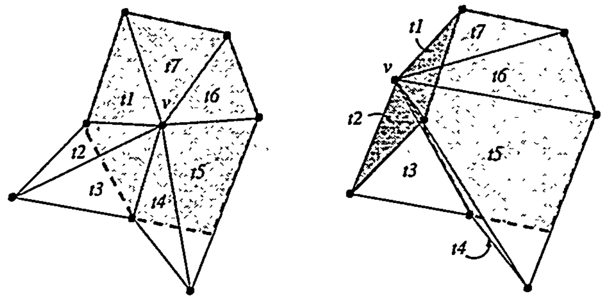

Figure 2: The submesh on the left is valid; the free vertex $v$ lies within the feasible region which is shaded gray. The submesh on the right is invalid; the free vertex $v$ lies outside the feasible region causing triangles $t1$ and $t2$ to have negative area.

routines and their input options are given in the Appendix.

## 2 Mesh Improvment Methods

Local mesh smoothing techniques are formulated in terms of the grid point to be adjusted, the *free vertex*, $v$, and that grid point's adjacent vertices, $V = adj(v) = \{v_i \ni$ an edge exists between $v$ and $v_i\}$. The location of the free vertex is changed according to some rule or heuristic procedure based on information available at the adjacent grid points. Suppose $\mathrm{x}$ is the position of the free vertex; then the general form of a local smoothing algorithm is given by

$$\mathrm{x}_{new} = \mathtt{Smooth}(\mathrm{x},\ V,\ |V|,\ conn(V)), \tag{1}$$

where $\mathrm{x}_{new}$ is the proposed new position of $v$, $|V|$ is the number of adjacent vertices, and $\mathrm{conn}(V)$ is the adjacent vertex connectivity information. Ideally, the new location of the free vertex will improve the mesh according to some measure of mesh quality such as dihedral angle or element aspect ratio. Let the values of mesh quality affected by a change in $\mathrm{x}$ be $\mathrm{f}(\mathrm{x}) = f_i(\mathrm{x})$, $i = 1, \ldots, n$. For example, if we use dihedral angle as a quality measure in a three-dimensional mesh, each tetrahedron would have six function values, one for each edge of the tetrahedron. Thus, the total number of function values affected by a change in $\mathrm{x}$ would be the number of tetrahedra containing the vertex $v$ multiplied by six. Let the minimum value of the functions evaluated at $\mathrm{x}$ be called the *active value*, and the set of functions that obtain that value, the *active set*, be denoted by $\mathcal{A}(\mathrm{x})$.

The action of the function `Smooth` is determined by the particular algorithm chosen. In this section the seven algorithms provided in Opt-MS are described.

### 2.1 Laplacian and "Smart" Laplacian Smoothing

For Laplacian smoothing, the action of the smoothing operator given in Equation (1) is to move the free vertex to the geometric center of the adjacent grid points. In this version of Laplacian smoothing, no effort is made to ensure that mesh quality is improved, and because the algorithm is heuristic, poor quality or even invalid elements can result from the use of this technique. See the two leftmost submeshs of Figure 5 for an example for which Laplacian smoothing fails. However, the technique is computationally inexpensive, easy to implement, and therefore commonly used.

A simple variant of Laplacian smoothing, which we call "smart" Laplacian smoothing, relocates the free vertex to the geometric center of the adjacent grid points only if the quality of the local mesh is improved according to some mesh quality measure. In this case, the smoothing operator given in Equation (1) has the following action:

```
Compute $\mathrm{f}(\mathrm{x}_0)$ and $\mathcal{A}(\mathrm{x}_0)$
Compute $\hat{\mathrm{x}} = \sum_{i \in V} \mathrm{x}_i/|V|$
Compute $\mathrm{f}(\hat{\mathrm{x}})$ and $\mathcal{A}(\hat{\mathrm{x}})$
```

If $\mathcal{A}(\hat{\mathbf{x}}) > \mathcal{A}(\mathbf{x}_0)$ set $\mathbf{x}_{new} = \hat{\mathbf{x}}$

where $\mathbf{x}_i$ is the position of the $i$th adjacent vertex. Computing $\hat{\mathbf{x}}$ is inexpensive, and the total time required by this method is dominated by the two function evaluations, $\mathbf{f}(\mathbf{x}_0)$ and $\mathbf{f}(\hat{\mathbf{x}})$.

## 2.2 Optimization-based Smoothing

The optimization approach used in Opt-MS finds the position $\mathbf{x}^*$ that maximizes the composite function

$$\phi(\mathbf{x}) = \min_{1 \leq i \leq n} f_i(\mathbf{x}). \tag{2}$$

As a particular example, if each $f_i$ were a triangle angle, then the optimization algorithm would seek to maximize the minimum angle in the local submesh. We illustrate the character of this function by showing a one-dimensional slice through a typical function $\phi$ in Figure 3. Note that each $f_i(\mathbf{x})$ is a smooth, continuously differentiable function and that multiple function values can obtain the minimum value. Hence, the composite function $\phi(\mathbf{x})$ has discontinuous partial derivatives where the active set $\mathcal{A}$ changes.

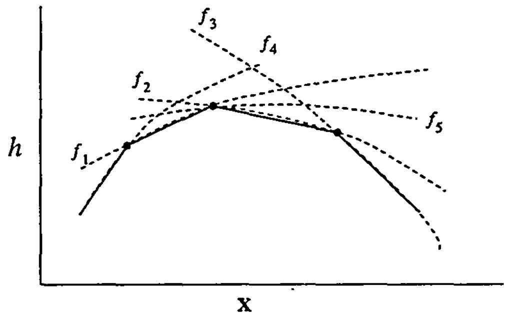

Figure 3: A one-dimensional slice through the nonsmooth function $\phi(\mathbf{x})$

We solve this nonsmooth optimization problem using an analogue of the steepest descent method for smooth functions. The search direction, s, at each step is the steepest descent direction from all possible convex linear combinations of the gradients in $\mathcal{A}(\mathbf{x})$. This is computed by solving the quadratic programming problem

$$\min \ \bar{\mathbf{g}}^T\bar{\mathbf{g}} \quad \text{where } \bar{\mathbf{g}} = \sum_{i \in \mathcal{A}} \beta_i \mathbf{g}_i(\mathbf{x})$$

$$\text{subject to } \sum_{i \in \mathcal{A}} \beta_i = 1, \ \ \beta_i \geq 0$$

for the $\beta_i$. The line search subproblem along s is solved by predicting the points at which the active set $\mathcal{A}$ will change. These points are found by computing the intersection of the projection of a current active function in the search direction with the linear approximations of each $f_i(\mathbf{x})$ given by the first-order Taylor series approximation. The distance to the nearest intersection point from the current location gives the initial step length, $\alpha$. The initial step is accepted if the actual improvement achieved by moving $v$ exceeds 90 percent of the estimated improvement or if the subsequent step results in a smaller function improvement. Otherwise $\alpha$ is halved recursively until a step is accepted or $\alpha$ falls below some minimum step length tolerance.

In general, the optimization process is terminated if one of the following conditions apply: (1) the step size falls below the minimum step length with no improvement obtained; (2) the maximum number of iterations is exceeded; (3) the achieved improvement of any given step is less than some user-defined tolerance; or (4) the Kuhn-Tucker conditions of nonlinear programming are satisfied indicating that we have found a local maximum $\mathbf{x}^*$ [5]. The action of the smoothing operator for optimization-based smoothing is given in Figure 4.

Amenta et al. [1] have shown that this technique is equivalent to generalized linear programming techniques by and thus the convex level set criterion can be used to determine whether there is a unique

```
$iter = 0$
Compute $f(x_0)$ and $\mathcal{A}(x_0)$
While $((x_{iter} \neq x^*)$ and $(\alpha >$ MIN_STEP$)$
        and $(iter <$ MAX_ITER$)$ and
        $(|\mathcal{A}(x_{iter}) - \mathcal{A}(x_{iter-1})| >$ MIN_IMP$))$
    Compute the gradients $g_{iter}$
    Compute search direction $s_{iter}$
    Compute $\alpha$
    While (STEP_NOT_ACCEPTED)
            and $(\alpha >$ MIN_STEP$)$
        Compute $x_{iter+1} = x_{iter} + \alpha s_{iter}$
        Compute $f(x_{iter+1})$ and $\mathcal{A}(x_{iter+1})$
        Test for step acceptance
        $\alpha = \alpha/2$
    Endwhile
    $iter = iter + 1$
Endwhile
```

Figure 4: The optimization-based smoothing algorithm

solution, $x^*$. Amenta et al. [1] described the level sets for several mesh quality criteria and found that many of them meet the convexity requirement for unique solutions in the feasible region. We note that similar local optimization-based smoothing methods have been proposed for a variety of optimization procedures and mesh quality measures, (see, e.g. Bank and Smith [3], Shephard and Georges [22], Staten et al. [21], and Knupp [18].

To illustrate a case for which optimization-based smoothing succeeds where Laplacian smoothing fails, we consider the initial local submesh drawn in the leftmost figure of Figure 5. In each submesh, the position of the free vertex is denoted with a black circle. The center submesh the shows the results of Laplacian smoothing; the free vertex position has moved outside the feasible region and elements $t2$ and $t3$ have become inverted. In the rightmost figure we show the results of optimization-based smoothing. The local submesh has significantly improved quality, and all of the elements remain valid.

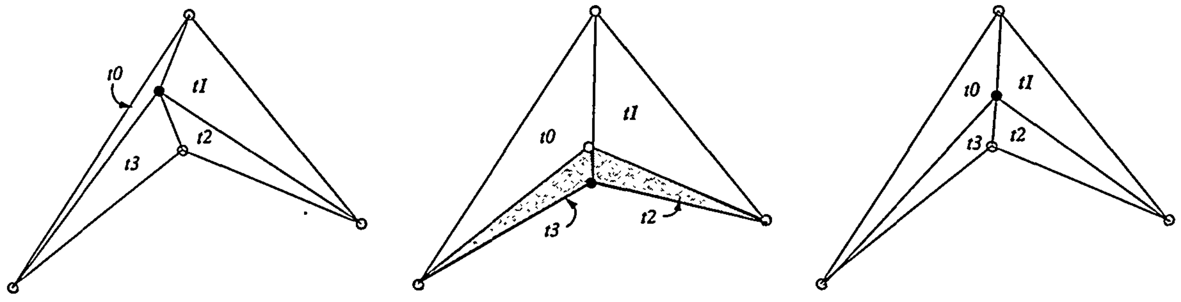

Figure 5: A local submesh for which optimization-based smoothing succeeds where Laplacian smoothing fails. The original local submesh is shown in the leftmost figure. The center figure shows the results of Laplacian smoothing, which is a tangled mesh. In the rightmost figure we show the results of optimization-based smoothing.

## 2.3 The Combined Approaches

A number of approaches that combine Laplacian and optimization-based smoothing can be used to improve the mesh as effectively as optimization-based smoothing at a fraction of the cost. In this section, we describe four such approaches available in the Opt-MS package.

**Combined Approach 1 (C1)** In this technique, the active value of the initial mesh is compared with a user-defined threshold value. If the threshold value is exceeded, the smart variant of Laplacian smoothing is used; otherwise, optimization-based smoothing is performed.

**Combined Approach 2 (C2)** In this technique, smart Laplacian smoothing is used as the first step for every grid point. The active value in the local mesh after this step is compared with a user-defined threshold value. If the active value exceeds the threshold value, the algorithm terminates; otherwise, optimization-based smoothing is performed.

**Combined Approach 3 (C3)** In this technique, the active value of the initial mesh is compared with a user-defined threshold value. If the threshold value is exceeded, no smoothing is performed; otherwise, Laplacian smoothing is used. If the active value still does not exceed the threshold value following Laplacian smoothing, the optimization-based smoother is used.

**Combined Approach 4 (C4) or Floating Threshold (F)** This technique is similar to the second combined approach except that we use a floating threshold rather than a fixed one. After each smoothing pass, the threshold value for the next pass is set equal to the global minimum active value in the mesh plus a constant. Experiments showed that a constant of five degrees works well in practice for both two and three dimensions.

## 2.4 Typical Results for Mesh Improvement

Typical results comparing the effectiveness and computational cost of the various smoothers for two- and three-dimensional application meshes are given in Freitag et al. [13], Freitag and Ollivier-Gooch [10], and Freitag [9]. In all cases, the mesh quality function used to determine the active value is the minimum sine of the angles in the incident elements. Effectiveness of the smoothing technique is measured by examining the global minimum and maximum angles/dihedral angles in two/three dimensions. Computational cost is measured by the average time required to smooth each vertex in the mesh. In each case studied in [9] the optimization-based method yielded a greater increase in the minimum angle than the Laplacian smoother did. The corresponding increase in computational cost is approximately a factor of four in two dimensions and a factor of ten in three dimensions. For all but one case, the combined approaches were able to obtain the same minimum angle as optimization-based smoothing used alone at a fraction of the cost.

In two dimensions, the cost of the combined approaches in decreasing order was C1, C2, C4, and C3, which corresponds to a decreasing number of function evaluations. In fact, the third combined approach often required less time than the Laplacian smoother because so few grid points required smoothing. In three dimensions, the ordering for cost effectiveness of the methods is mixed, but the C4 approach always obtains a good mesh at a low computational cost. The cost of the third approach can be further reduced by evaluating only grid points that were repositioned or are adjacent to grid points that were repositioned in the previous smoothing pass.

In addition, the C1, C2, and C4 approaches created more equilateral elements than optimization-based smoothing in both two and three dimensions. Thus, these techniques tend to generate higher-quality meshes than either Laplacian or optimization-based smoothing used alone and their use is recommended; particularly C2 and C4. In contrast, the C3 approach had a relatively poor angle distribution because only a small number of the grid points were relocated. Its use is recommended only for the cases in which the time to smooth the mesh must be small and the user is interested only in eliminating extremely distorted elements rather than an overall improvement to the mesh.

## 2.5 Quality Metrics Provided in Opt-MS

The Opt-MS package includes a number of geometry-based quality metrics for two- and three-dimensional meshes. Many of these measures are angle based, but new metrics can be added upon request.[1] In addition,

[1] Currently supported metrics are targeted at improving meshes for isotropic problems. Anisotropic smoothing typically requires application solution information in addition to geometric information and is therefore problem dependent. The

future releases of Opt-MS will allow the users to input their own quality metric function and gradient routines for optimization-based smoothing.

Table 1 contains a list of currently supported metrics thatmeasure triangle quality. The Opt-MS reference variable, the number of function evaluations per triangle, $n$, and the formula used to compute the function value are also given. In the formulas given, $\theta_i$ is an angle of the triangle, $A_t$ is the area, $J_t$ is the Jacobian (two times the area), $J_e$ is the Jacobian of an equilateral triangle defined by the edge opposite the free vertex, and $L_i$ is the length of the edge opposite $\theta_i$.

Table 1: Two-Dimensional Quality Metrics

| Quality Metric | Opt-MS Variable | $n$ | $f_i$ |
|---|---|---|---|
| Maximize the minimum angle | MAX_MIN_ANGLE | 3 | $\theta_i$ |
| Minimize the maximum angle | MIN_MAX_ANGLE | 3 | $-\theta_i$ |
| Maximize the minimum cosine | MAX_MIN_COSINE | 3 | $\cos(\theta_i)$ |
| Minimize the maximum cosine | MIN_MAX_COSINE | 3 | $-\cos(\theta_i)$ |
| Maximize the minimum sine | MAX_MIN_SINE | 3 | $\sin(\theta_i)$ |
| Minimize the maximum deviation from equilateral | MIN_MAX_JACOBIAN_DIFF | 1 | $-\frac{(J_t-J_e)^2}{J_e}$ |
| Maximize the minimum scaled Jacobian | MAX_MIN_SCALED_JACOBIAN | 3 | $\frac{J_t}{L_jL_k}$ |
| Maximize the minimum ratio of triangle area to edge length | MAX_MIN_AREA_LENGTH_RATIO | 1 | $\frac{12A_t}{\sqrt{3}(L_1+L_2+L_3)}$ |
| Default (max min sine) | FUNCTION_DEFAULT_2D | 3 | $\sin(\theta_i)$ |

Table 2 contains a list of the currently supported metrics that measure tetrahedron quality. The Opt-MS input variable, the number of function evaluations per tetrahedron, $n$, and the formula used to compute the function value are also given. In the formulas given in Table 2, $\theta_i$ is a dihedral angle of the tetrahedron, $V_t$ is the volume of the tetrahedron, $J_t$ is the Jacobian (six times the volume), and $L_i$ is the length of the edge opposite $\theta_i$.

Table 2: Three-Dimensional Quality Metrics

| Quality Metric | Opt-MS Variable | $n$ | $f_i$ |
|---|---|---|---|
| Maximize the minimum dihedral angle | MAX_MIN_DIHEDRAL | 6 | $\theta_i$ |
| Minimize the maximum dihedral angle | MIN_MAX_DIHEDRAL | 6 | $-\theta_i$ |
| Maximize the minimum cosine of dihedral angle | MAX_MIN_COSINE_DIHEDRAL | 6 | $\cos(\theta_i)$ |
| Minimize the maximum cosine of dihedral angle | MIN_MAX_COSINE_DIHEDRAL | 6 | $-\cos(\theta_i)$ |
| Maximize the minimum sine of dihedral angle | MAX_MIN_SINE_DIHEDRAL | 6 | $\sin(\theta_i)$ |
| Default (max minimum sine of dihedral angle) | FUNCTION_DEFAULT_3D | 6 | $\sin(\theta_i)$ |

Two pairs of these measures give nearly identical results, though not necessarily identical convergence behavior:

$$\begin{aligned} \text{maxmin angle} &\approx \text{minmax cosine} \\ \text{minmax angle} &\approx \text{maxmin cosine.} \end{aligned}$$

In addition, test cases showed that metrics that seek only to remove large angles from the mesh do not succeed in eliminating small angles; however the criteria that eliminate small angles also succeed in eliminating large angles [10]. Maximizing the minimum sine of the (dihedral) angles successfully eliminates poor angles at both extremes because it is small near the angles of 0° and 180°. This is the default metric in both two and three dimensions.

**interfaces necessary to support this functionality will be included in future releases of the Opt-MS software.**

## 3 Mesh Untangling

The mesh untangling method provided in Opt-MS is a local technique formulated in the same manner as the smoothing techniques. That is, given a free vertex, $v$, and its adjacent vertices $V = adj(v)$, the new position of the free vertex, $\mathbf{x}_{new}$, is given by the general operation

$$\mathbf{x}_{new} = \texttt{Untangle}(\mathbf{x},\, V,\, |V|,\, \text{conn}(V)).$$

Ideally, $\mathbf{x}_{new}$ will either untangle the local submesh, or improve the local submesh in such a way that it can be untangled in a succeeding sweep through the mesh. The action of the operator `Untangle` can take a variety of forms ranging from heuristic procedures similar to Laplacian smoothing to optimization techniques such as those described earlier. In this section, we describe a formulation based on linear programming techniques that is guaranteed to converge for each local submesh problem.[2]

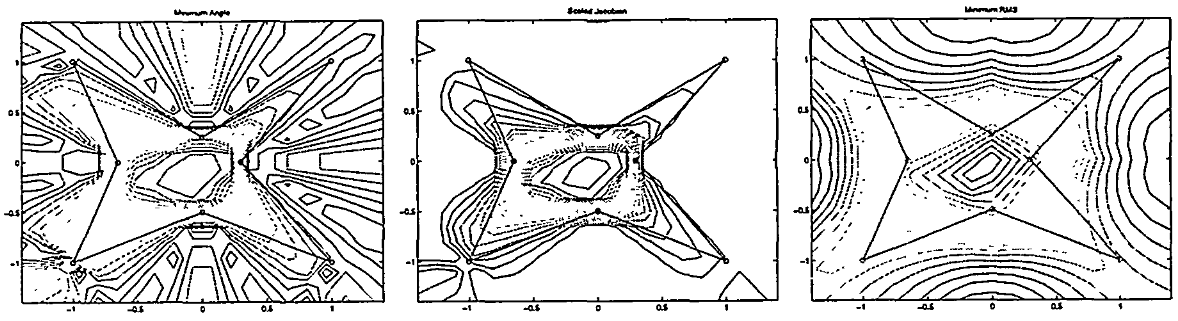

Figure 6: Level sets for the minimum angle, minimum sine of an angle, and minimum root mean square quality metrics. Each of these metrics is convex in the feasible region defined by the interior convex hull of the local submesh, but is nonconvex outside the feasible region.

### 3.1 Formulation and Solution

To guarantee convergence on the local submesh, the function level sets must be convex regardless of the position of the free vertex [1]. Level sets for several of the mesh quality metrics discussed in Section 2 are shown in Figure 6. These metrics are clearly nonconvex if the free vertex lies outside of the feasible region, and convergence cannot be guaranteed for these measures when the mesh is invalid. In fact, preliminary tests using these metrics for mesh untangling often failed to converge.

One function that has convex level sets regardless of the position of the free vertex is minimum element area (volume in 3D) in a local submesh

$$f(\mathbf{x}) = \min_{1 \le i \le n} A_i(\mathbf{x}),$$

where $n$ is the number of simplices in the local submesh, $A_i$ is the area (volume) of simplex $t_i$, and $\mathbf{x}$ is the position of the free vertex. In two dimensions, if triangle $t_i$ is defined by the free vertex position, $\mathbf{x}$, and the positions of two other vertices, $\mathbf{x}_i$ and $\mathbf{x}_j$, then $A_i$ can be expressed as a function of the Jacobian of the element [18]

$$A_i = \det(\mathbf{x}_i - \mathbf{x},\ \mathbf{x}_j - \mathbf{x}) = a_{x_i} x + a_{y_i} y + c_i,$$

where

$$a_{x_i} = y_i - y_j, \qquad a_{y_i} = x_j - x_i, \qquad c_i = x_i y_j - x_j y_i.$$

Similarly in three dimensions, if tetrahedron $t_i$ is defined by the free vertex position, $\mathbf{x}$ and the positions of three other vertices, $\mathbf{x}_i$, $\mathbf{x}_j$, and $\mathbf{x}_k$, then $A_i$ is given by

$$A_i = \det(\mathbf{x}_i - \mathbf{x},\ \mathbf{x}_j - \mathbf{x},\ \mathbf{x}_k - \mathbf{x}) = a_{x_i} x + a_{y_i} y + a_{z_i} z + c_i,$$

[2]The global convergence properties of the local untangling method are not yet well understood, although the method works well in both two and three dimensions.

where

$$a_{xi} = -\det\begin{bmatrix} 1 & 1 & 1 \\ y_i & y_j & y_k \\ z_i & z_j & z_k \end{bmatrix} \quad a_{y_i} = -\det\begin{bmatrix} x_i & x_j & x_k \\ 1 & 1 & 1 \\ z_i & z_j & z_k \end{bmatrix} \quad a_{zi} = -\det\begin{bmatrix} x_i & x_j & x_k \\ y_i & y_j & y_k \\ 1 & 1 & 1 \end{bmatrix}$$

$$c_i = \det\begin{bmatrix} x_i & x_j & x_k \\ y_i & y_j & y_k \\ z_i & z_j & z_k \end{bmatrix}$$

The associated optimization problem for mesh untangling is then

$$\max \min_{1\le i\le n} A_i(\mathrm{x}). \tag{3}$$

An example of the level sets typical of this function are shown in Figure 7 for three different local submeshes, including one that has fixed edges that are tangled. We note that this function is not suitable for optimization-based mesh improvement because a small, but perfectly shaped element is likely to be distorted in an effort to maximize its area.

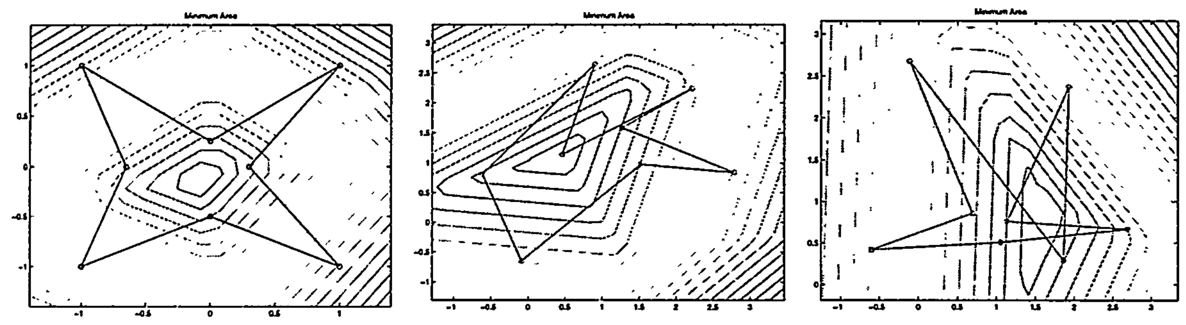

Figure 7: Level sets for the minimum element area function for three local submeshes, including one that has fixed edges that are tangled

We showed that the level sets are convex in both two and three dimensions in [12]. In two dimensions, the level sets are given by

$$f(\mathrm{x}) = \min_{1\le i\le n} A_i(\mathrm{x}) \;=\; \min_{1\le i\le n} \frac{1}{2} b_i h_{\perp i}(\mathrm{x}) \;=\; C, \tag{4}$$

where $C$ is a constant, $b_i$ is the length of the fixed edge of the triangle $i$, and $h_{\perp i}$ is the perpendicular distance from x to the line defined by the fixed edge of triangle $i$. Note that there are no restrictions on $C$; it can be greater than, equal to, or less than zero. Equation 4 shows that all points on a line parallel to the fixed edge of the triangle create triangles of equal area. Thus, for a local submesh, the level sets are the intersection of the half planes defined by the lines an equal distance and parallel to the fixed edge of each triangle. These half planes are convex regions; the intersection of convex regions is convex, and therefore the level sets are convex. An analogous argument can be made for three-dimensional local submeshes.

In both 2D and 3D, $A_i$ is a linear function of the free vertex position, x. Thus Equation 3 is a linear programming problem that is formulated as follows. Let $d$ be the spatial dimension of the problem and $n$ be the number of incident elements. Define the $(d+1)\times n$ matrix $\mathcal{A}$ to be the matrix whose $i$th column contains the entries $a_{xi}$, $a_{y_i}$, 1 for $d=2$, and $a_{xi}$, $a_{y_i}$, $a_{zi}$, 1 for $d=3$ and $\pi$ to be the $(d+1)$-vector containing the spatial coordinates of the free vertex in the first $d$ components and the current minimum area (volume) in the last component. Then, by definition of $\mathcal{A}$ and $\pi$,

$$\mathcal{A}^T\pi = \mathrm{c} - \mathrm{s},$$

where c is the $n$-vector containing the values of $c_i$ defined above, and s is an $n$-vector of slack variables, where the $i$th component, $s_i$, gives the difference between the area (volume) of simplex $t_i$ and the current minimum area (volume). Thus, the dual of the linear programming problem is

$$\begin{aligned} \max \quad & \mathrm{b}^T \pi \\ \text{subject to} \quad & \mathcal{A}^T \pi + \mathrm{s} = \mathrm{c}, \quad \mathrm{s} \geq 0, \end{aligned}$$

where b is a $(d+1)$-vector whose first $d$ components are zero and whose last component is one, so that $\mathrm{b}^T\pi$ gives the current minimum simplex area (volume).

The primal of the linear program is then

$$\min \quad \mathrm{c}^T \mathrm{y} \tag{5}$$

$$\text{subject to} \quad \mathcal{A}\mathrm{y} = \mathrm{b}, \quad \mathrm{y} > 0 \tag{6}$$

where y is the primal solution vector.

The linear programming problem defined by equations (5) and (6) can be solved using the simplex method [14]. The phase one solution can also be formulated as a linear programming problem; the details of the formulation and solution can be found in [12]. The linear program has been solved when $s_i \geq 0$, $i = 1 \ldots n$ and the complementarity condition $\mathrm{y}^T\mathrm{s} = 0$ has been satisfied.

## 3.2 Typical Results for Mesh Untangling

We illustrated the fact that local submeshes exist for which Laplacian smoothing will fail in Figure 5. In Figure 8, we show the same local submesh but with the initial position of the free vertex outside the feasible region. The second submesh is the result of Laplacian smoothing, which is also a tangled submesh. In the last figure, the result of using the optimization-based approach to mesh untangling is shown. In this case the free vertex is moved to a location that results in a positive area for all incident elements.

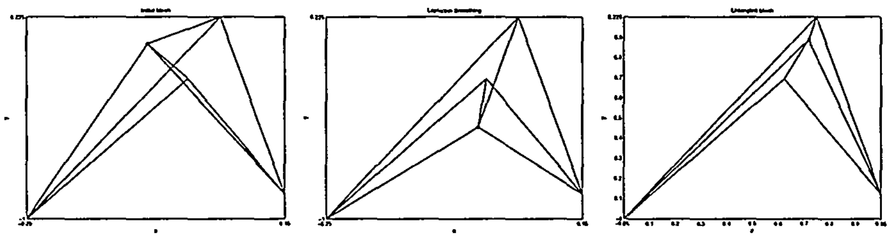

Figure 8: A local submesh that shows that Laplacian smoothing can sometimes fail. A tangled, local submesh is shown in the leftmost figure. The center figure shows the results of Laplacian smoothing, which is also a tangled local submesh. In the right figure, we show the results of mesh untangling using the linear programming approach.

To show typical results for the mesh untangling problem, we start with a two-dimensional Delaunay mesh created using the Triangle package [23]. This mesh is perturbed in a random fashion so that a user-defined percentage of the nodes are moved a user-defined distance. The leftmost mesh in Figure 9 shows the case in which 10 percent of the elements of mesh are perturbed a distance equal to the average element edge length, $h$. The perturbed mesh is untangled by using the linear programming approach. The untangling process stops when all of the elements are valid or a maximum number of sweeps have been performed (in this case twenty). The untangling proceedure results in meshes of extremely poor quality; minimum angles of $10^{-5}$ degrees are typical. Therefore we follow mesh untangling with three passes of optimization-based smoothing using the second combined approach described in Section 2 and the quality metric "maximize the minimum sine". In Figure 9, the central mesh shows the results of mesh untangling, and the rightmost figure shows the same mesh after three passes of smoothing.

To examine the effectiveness of mesh untangling as the number and severity of the tangled elements increase, we created two series of meshes. In the first series the magnitude of the perturbation is fixed to be the average element edge length, $h$, and additional nodes are perturbed in each successive mesh in the series. In the second series of meshes, the number of nodes that are perturbed is held fixed, but the magnitude of the perturbation is increased in each successive mesh. The results showed that the amount a grid point is

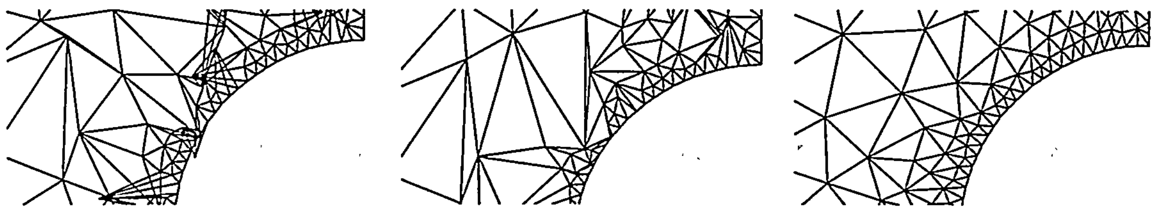

Figure 9: Typical results for mesh untangling using the linear programming approach. The mesh on the left is the original, tangled mesh; the mesh in the middle is the same mesh after untangling, and the mesh on the right is the same mesh after mesh smoothing.

perturbed significantly increases the number of untangling passes required and decreases the effectiveness of three passes of mesh smoothing. In contrast, the number of invalid elements did not significantly effect the number sweeps required to untangle the mesh. The cost of mesh untangling per grid point is three times smaller than the cost of mesh smoothing using the C2 approach.

## 4 Using the Opt-MS Package

In this section, we describe the use of Opt-MS package to improve an application mesh. In particular, we discuss minimal amount of code needed to smooth an application mesh, give a detailed example of the user input required for each local submesh, describe the usage of the mesh quality assessment routines, and show typical output from the Opt-MS statistics and profiling options. The details of the API function calls and their input options can be found in the appendix; the details of the algorithms can be found in Sections 2 and 3. The software is written in C, and the routines have been tested with C and C++ applications. Fortran bindings will be provided in a future release.

### 4.1 Smoothing and Untangling Local Submeshes

The primary functionality of the Opt-MS package is the smoothing and untangling of local simplicial submeshes (triangles in 2D, tetrahedra in 3D). This functionality can be accessed by using three Opt-MS library functions:

- `SMinitSmoothing` to create and initialize the Opt-MS data structures. This routine must be called once before any other Opt-MS calls are made. Input to this routine allows the user to specify options for the quality metric, the optimization technique, and threshold value for the combined approaches. This routine returns a `void *` data structure that must be passed to most other Opt-MS functions.
- `SMsmooth` or `SMuntangle` to adjust the position of a free vertex. These routines require several input arguments from the user, which are described in detail in Section 4.2. One of these arguments is the spatial location of the free vertex; on output, this argument is overwritten with the new position of the free vertex.
- `SMfinalize` to free the data structure created by SMinitSmoothing. This routine should be called once when mesh optimization is complete.

The skeleton program given below illustrates the basic use of these three calls. The OptMS.h file must be included in the preamble to define the smoothing functions and Opt-MS constants. The `SMinitSmoothing` function is called once before looping through the interior vertices of the mesh and smoothing them. The options available to the user for initializing the smoothing technique, mesh quality metric, problem dimension, etc, are described in detail in the appendix. In this example, a single smoothing pass through the mesh vertices is made before the function `SMfinalize` is called to free the smoothing data structures `smooth_data`. Mesh untangling can be performed by simply replacing the `SMsmooth` call with a call to `SMuntangle`, the argument lists are the same.

```
/*------------- Preamble-------------------- */
#include "OptMS.h"

/*------------- Main Routine---------------- */
int main(int argc, char **argv)
{
  int                i;
  int                dimension = 2;      /* the problem dimension */
  Mesh_data          *mesh;              /* the user's mesh data structure */
  void               *smooth_data;       /* the smoothing data structure */
  int                num_incident_vtx;   /* number of incident vertices */
  int                num_incident_tri;   /* number of incident triangles */
  int                free_vtx[2];        /* coordinates of the free vertex */
  double             **vtx_list;         /* coordinates of the incident vtx */
  int                **vtx_connectivity; /* connectivity of the incident tri */

  /*------------- Initialize the user's mesh -----------*/
  initMesh(&mesh);   /* a user written routine */

  /*----- Initialize the Opt-MS data structures ---- */
  /* If argc and argv are not available, NULL may be passed in their place */
  SMinitSmoothing(argc, argv, dimension, OPTMS_COMBINED, MAX_MIN_SINE,
                             OPTMS_DEFAULT,  &smooth_data);

  /*------------ Smooth the mesh------------ */
  for (i=0;i<mesh->num_nodes;i++) {
    if (!mesh->vtx[i]->boundary) {


      /*---------------------------------------------------------------
      This is a user-written subroutine to fill the data structures
      necessary for smoothing a local submesh.  These data structures
      are described in detail in the next subsection
      ---------------------------------------------------------------*/
      getLocalSubmesh(mesh, mesh->vtx[i], num_incident_vtx, num_incident_tri,
                      free_vtx, vtx_list, vtx_connectivity);

      /*---------------- Smooth a local submesh--------------------- */
      SMsmooth(num_incident_vtx,num_incident_tri,free_vtx,
               vtx_list,vtx_connectivity,smooth_data);
    }
  }

  /*------- Release the Opt-MS data structures------*/
  SMfinalizeSmoothing(smooth_data);
}
```

There are several API functions that allow the user to change the technique (SMsetSmoothTechnique), quality metric (SMsetSmoothFunction), threshold value (SMsetSmoothThreshold), or problem dimension (SMsetProblemDimension) at any time during the mesh optimization process. These are described in more detail in the appendix.

## 4.2 Input to SMsmooth and SMuntangle

The `SMsmooth` and `SMuntangle` subroutines require spatial and connectivity information about the local submesh, which the user must provide in the following argument list:

```
SMsmooth(int num_incident_vtx, int num_incident_tri,
         double *free_vtx, double **incident_vtx,
         int **vtx_connectivity, void *smooth_data).
```

### 4.2.1 Two-Dimensional Submeshes

To illustrate data that fills the argument list for a two-dimensional local submesh, we consider the submesh shown in Figure 10. In this case, the free vertex, $v$, is connected to five incident vertices, $v0$, $v1, \ldots,$ $v4$ with spatial positions as shown, and five incident triangles, $t0$, $t1, \ldots,$ $t4$.

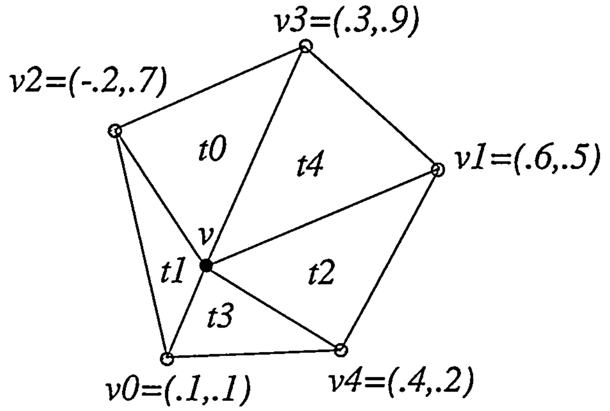

Figure 10: A typical two-dimensional local submesh in which $v$ is the free vertex, $v0$, $v1, \ldots,$ $v4$ are the incident vertices, and $t0$, $t1, \ldots,$ $t4$ are the incident triangles.

If the current position of $v$ is (.2,.25), then the input arguments to the function `SMsmooth` or `SMuntangle` are as follows:

- `num_incident_vtx`: an integer containing the number of vertices adjacent to the free vertex, in this case 5
- `num_incident_tri`: an integer containing the number of triangles incident to the free vertex, in this case 5
- `free_vtx`: a pointer to a two-vector of type double containing the $x$ and $y$ coordinates of the free vertex, in this case free_vtx[0] = .2 and free_vtx[1] = .25.
- `incident_vtx`: a pointer to an array of two-vectors of type double containing the spatial positions of the incident vertices, in this case

| | |
|---|---|
| incident_vtx[0][0] = .1 | incident_vtx[0][1] = .1 |
| incident_vtx[1][0] = .6 | incident_vtx[1][1] = .5 |
| incident_vtx[2][0] =-.2 | incident_vtx[2][1] = .7 |
| incident_vtx[3][0] = .3 | incident_vtx[3][1] = .9 |
| incident_vtx[4][0] = .4 | incident_vtx[4][1] = .2 |

- `vtx_connectivity`: a pointer to an array of two-vectors of type int containing the connectivity of the incident triangles. The triangles must be right handed with the free vertex ordered first. The integer values correspond to the index location of the incident triangle vertices in the `incident_vtx` array given above. In this case, the connectivity information for elements $t0, \ldots, t4$ are

| | |
|---|---|
| vtx_connectivity[0][0] = 3 | vtx_connectivity[0][1] = 2 |
| vtx_connectivity[1][0] = 2 | vtx_connectivity[1][1] = 0 |
| vtx_connectivity[2][0] = 4 | vtx_connectivity[2][1] = 1 |
| vtx_connectivity[3][0] = 0 | vtx_connectivity[3][1] = 4 |
| vtx_connectivity[4][0] = 1 | vtx_connectivity[4][1] = 3 |

- `smooth_data`: a void pointer the data structure created in SMinitSmoothing

### 4.2.2 Three-Dimensional Submeshes

An example for the input required in three dimensions is given for the partial three-dimensional local submesh shown in Figure 11. Only two of the incident tetrahedron are shown; a full local submesh will typically contain many more incident elements than illustrated here.

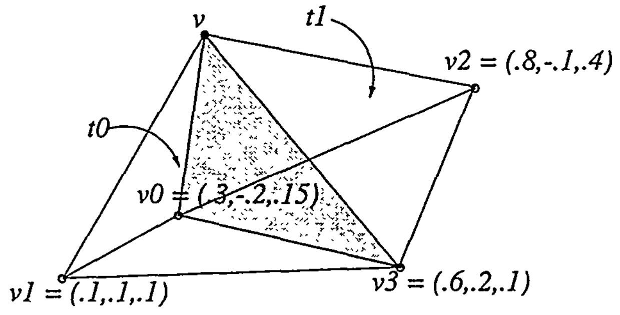

Figure 11: A partial three-dimensional local submesh in which $v$ is the free vertex, $v0, \ldots, v3$ are the incident vertices, and $t0$ and $t1$ are two of the incident tetrahedron.

If the current position of $v$ is (.35,.25,.6) and the number of incident vertices is $n$, then the input arguments to the function `SMsmooth` or `SMuntangle` are as follows:

- `num_incident_vtx`: an integer containing the number of vertices adjacent to the free vertex, in this case $n$
- `num_incident_tri`: an integer containing the number of triangles incident to the free vertex, in this case $2n - 4$
- `free_vtx`: a pointer to a three-vector of type double containing the $x$, $y$, and $z$ coordinates of the free vertex, in this case free_vtx[0] = .35, free_vtx[1] = .25, and free_vtx[2] = .6
- `incident_vtx`: a pointer to an array of three-vectors of type double containing the spatial positions of the incident vertices. For the four incident vertices illustrated in Figure 11, the entries are

| | | |
|---|---|---|
| incident_vtx[0][0] = .3 | incident_vtx[0][1] = -.2 | incident_vtx[0][2] = .15 |
| incident_vtx[1][0] = .1 | incident_vtx[1][1] = .1 | incident_vtx[1][2] = .1 |
| incident_vtx[2][0] = .8 | incident_vtx[2][1] = -.1 | incident_vtx[2][2] = .4 |
| incident_vtx[3][0] = .6 | incident_vtx[3][1] = .2 | incident_vtx[3][2] = .1 |

- `vtx_connectivity`: a pointer to an array of three-vectors of type int containing the connectivity of the incident tetrahedra. The tetrahedra must be right-handed with the free vertex ordered first. The integer values correspond to the index location of the incident tetrahedra vertices in the `incident_vtx` array given above, in this case

| | | |
|---|---|---|
| vtx_connectivity[0][0] = 0 | vtx_connectivity[0][1] = 1 | vtx_connectivity[0][2] = 3 |
| vtx_connectivity[1][0] = 3 | vtx_connectivity[1][1] = 2 | vtx_connectivity[0][2] = 0 |

- `smooth_data`: a void pointer the data structure created in SMinitSmoothing

## 4.3 Measuring Mesh Quality

The user may obtain information regarding mesh quality by looping through the elements of the mesh and accumulating quality information. This process involves calling three Opt-MS subroutines to initialize the quality table, SMinitQualityTable, accumulate information about the quality of individual elements, SMaccumulateQualityInformation, and then print the accumulated information, SMprintQualityInformation. The input argument to the initialization and print functions is the smoothing data structure created by SMinitSmoothing, and input to the accumulation routine is that same data structure together with the spatial location of the vertices of the element simplex in right-hand order as a double ** array. The vertices array will be of dimension $3 \times 2$ for triangles and of dimension $4 \times 3$ for tetrahedron.

Example usage of the quality assessment routines is provided below for a two-dimensional triangular mesh. A three-dimensional example using these routines can be found in the file OPTMS_DIR/examples/3d/Smooth.c. Note that the quality table must be reset before each pass through the mesh or information will continue to accumulate over multiple passes.

```
/* Declaratives */
double vertices[3][2];

 ... user code... initialize the mesh...  SMinitSmoothing(...)...

/***************************************************************/
/* Assess the quality of the mesh                              */
/***************************************************************/
SMinitQualityTable(smooth_data);
for (i=0;i<mesh->num_tri;i++) {
   for (j=0;j<3;j++) {
     vertices[j][0] = mesh->vtx[vertex_id]->coord[0];
     vertices[j][1] = mesh->vtx[vertex_id]->coord[1];
   }
   SMaccumulateQualityInformation(smooth_data,vertices);
}
SMprintQualityInformation(smooth_data);

....
```

The call to SMprintQualityInformation will result in a table of mesh quality information printed to stdout as shown below. Quality metrics that are currently accumulated are the minimum, maximum, and average values of element angles (dihedral angles in 3D), the deviation of the element from an equilateral element, the scaled Jacobian, and, the element area. The number of values accumulated for each function is contained at the end of each row in parentheses. If more than one value is accumulated for each element, the average minimum and maximum values are also reported.

```
Mesh Quality Information for 878 Elements
-------------------------------------------------------------------------------------
Quality Metric (Target)           Min Value  Max Value  Avg Value  Avg Min    Avg Max
-------------------------------------------------------------------------------------
              Angle (6.00e+01)    2.62e+01   1.28e+02   6.00e+01   4.05e+01   8.57e+01 (2634)
Deviation from Equil (0.00e+00)  -1.82e+00  -6.24e-06  -1.51e-01  -1.51e-01  -1.51e-01 (878)
     Scaled Jacobian (8.60e-01)   4.41e-01   1.00e+00   8.01e-01   6.43e-01   9.64e-01 (2634)
       Triangle Area (0.00e+00)   8.16e-05   7.68e-03   1.14e-03   1.14e-03   1.14e-03 (878)
-----------------------------------------------------------
There are 0 invalid elements in the mesh
```

In addition, the number of invalid elements found in the mesh, those with negative area (or volume), is also reported. The quality information accumulated about the mesh can be used in conjunction with mesh untangling by using the function call SMinvalidMesh, which returns TRUE if invalid elements were found in the last accumulation of quality information. Hence one could write a loop to untangle a mesh as follows:

```
while (SMinvalidMesh(smooth_data) && (untangle_pass < MAX_PASSES)) {
    Loop over the nodes, untangling the mesh
    SMinitQualityTable
    Loop over the elements, evaluating the quality of the mesh
    untangle_pass++
}
```

## 4.4 Gathering and Reporting Opt-MS Statistics.

To accumulate and print statistics about mesh improvement, the Opt-MS library should be configured with the -enable-stats option. If this option is enabled, statistics are automatically accumulated for each local submesh. Minimal impact on performance is seen when this functionality is enabled; there is no degradation of performance if the functionality is not enabled. To access the information that is accumulated, the user must call two additional Opt-MS subroutines each sweep through the mesh as follows:

```
 for (i=0;i<NUM_SMOOTH_PASSES;i++) {
    SMinitSmoothStats(smooth_data);
    Loop over the nodes, smoothing the mesh
    SMprintSmoothStats(smooth_data);
}
```

The SMinitSmoothStats call reinitializes the statistics data structure so that statistics for each pass through the mesh can be accumulated individually, and the SMprintSmoothStats call prints the following table of information for the accumulated data.

```
******************************************************************
SMOOTHING STATISTICS
******************************************************************
The approximate global minimum value                        0.006509
The total number of nodes smoothed                          400
The number of calls to Laplacian smoothing                  400
     Invalid Laplacian steps (percent)                      94  (23.500000)
     Non-improvement Laplacian steps (percent)              60  (15.000000)
The number of calls to optimization smoothing               389
     Average number of iterations/optimization call         5.961440
The number of cells with no improvement                     1
The average final active value                              0.244698
The average improvement (over all cells)                    0.140318
The termination status:
  Laplacian smoothing enough (percentage)     11 (2.750000)
  Equilibrium pt found (percentage)           319 (79.750000)
     Started at equilibrium (percentage)      0 (0.000000)
  Zero search direction (percentage)          1 (0.250000)
  Improvement too small (percentage)          64 (16.000000)
  Flat no improvement (percentage)            5 (1.250000)
  Step too small (percentage)                 0 (0.000000)
  Max Iter Exceeded (percentage)              0 (0.000000)
******************************************************************
```

The first value is an approximation to the global minimum value in the mesh. Because this value is determined by comparing the local minimum value with previously completed local submesh minimum values, this value is not necessarily the true global minimum value. That is, because mesh elements are included in more than one local submesh, poor-quality elements may be improved more than once in different local submeshes. However, the statistics-gathering routine works only with local submeshes and cannot track this global information. Thus, only the first improvement will be recorded and the true minimum value may be greater than the one reported. The routine also prints statistics on the number of nodes for which Laplacian smoothing was used, and how many of those cases resulted in an invalid step or yielded no improvement to the mesh and were therefore not accepted. The optimization statistics include the number of nodes smoothed by using optimization-based technique, the average number of optimization iterations needed to adjust each grid point, the average active value, and the average improvement in the local submeshes. In addition, the termination status percentages for the local submeshes are reported.

## 4.5 Profiling Opt-MS

The Opt-MS package has been instrumented by using the SUMAA_LOG system to monitor the time required to complete various major events in the smoothing code.[3] The logging library is designed to be lightweight and nonintrusive when enabled. When the functionality is not enabled, there is no degradation of performance.

This functionality is enabled by configuring the library with -enable-logging option. The monitored events in the smoothing code include the main call to `SMsmooth`, which gives the total time for smoothing the local submesh. That time is further divided into optimization and Laplacian totals and important subcomponent totals such as function and gradient evaluation and search direction computation. For each event, the number of calls made is reported, along with the total time, time per call, and percentage of the total time spent in each event. These calls are nested so that the time spent in children routines are reported as a part of the parent event's time. Sample output from Opt-MS profiling is shown below. If the argc and argv variables from the main routine are passed to `SMinitSmoothing`, the command line option `-log_file filename` can be used to print logging information to a file. If this option is not desired, or argc and argv are not available, the user may pass NULL arguments instead of argc and argv.

```
Log Performance Summary (Single Processor) :---------------------------------
Total Time (sec):   2.450e-01
Total Flops:        0.000e+00
MegaFlops:          0.000e+00
-----------------------------------------------------------------------------
No flops accumulated, printing -1 in percent flops column

Event Summary:-------------------------------------------------------------
   Count: number of times phase was executed
   Time and Flops/sec:
      Max - maximum over all the processors
      Ratio - ratio of max to min over all processors
   Global: entire computation
      %T - percent time in this event
      %F - percent flops in this event
-----------------------------------------------------------------------------
Phase            Count   Total Time(s)  Time/call   Flops     %Time   %Flops
-----------------------------------------------------------------------------
Smooth           1200         2.45e-01    2.04e-04   0.00e+00  100.0    -1.0
Init Smooth      1200         5.65e-03    4.71e-06   0.00e+00    2.3   -1.0
Initialize       2401         3.50e-02    1.46e-05   0.00e+00   14.3   -1.0
Optimize          558         2.01e-01    3.60e-04   0.00e+00   82.0   -1.0
Function         4496         4.12e-02    9.16e-06   0.00e+00   16.8   -1.0
Gradient         1412         1.23e-01    8.73e-05   0.00e+00   50.3   -1.0
Search Dir       1412         6.16e-03    4.36e-06   0.00e+00    2.5   -1.0
```

[3]This system allows the user to monitor both the time and flop counts for user-defined events in the code, but in the current release, only the time used is monitored.

```
Edge/Face Srch     90          3.19e-04     3.54e-06   0.00e+00   0.1     -1.0
End Smooth       1200          7.49e-04     6.24e-07   0.00e+00   0.3     -1.0
Init Stats          3          3.93e-06     1.31e-06   0.00e+00   0.0     -1.0
Lap Smooth       1200          1.87e-02     1.56e-05   0.00e+00   7.6     -1.0
```

## 5 Compiling and Linking Opt-MS

The Opt-MS library is distributed with C source code that can be configured and built to meet a particular user's needs. In this section, the directory structure is described as well as the commands and options available for configuring and compiling the library. The use of the library is described in detail in Section 4.

### 5.1 Opt-MS Directory Structure

In the documentation that follows, it is assumed that the user has successfully downloaded and extracted the Opt-MS package from the file `Opt-MS.tar.Z` using the commands

```
uncompress Opt-MS.tar.Z; tar -xvf Opt-MS.tar
```

Let OPTMS_DIR be the local directory into which the package was extracted, for example `/home/me/Opt-MS`.

The directory structure of the Opt-MS distribution is as follows:

- docs: contains the documentation for the Opt-MS library in both postscript and html format in the subdirectories tex and www, respectively.
- examples: contains the two- and three-dimensional examples in the subdirectories 2d and 3d, respectively. Each subdirectory contains a driver code, `Smooth.c`, that demonstrates most of the functionality currently available in the Opt-MS package. In particular, each driver can initialize a number of test meshes, assess their quality, untangle meshes with invalid elements, and smooth the resulting valid mesh.
- include: contains the include files for the library, including the file OptMS.h that must be included in the preamble of a user's code.
- lib: created during the installation process and contains the versions of the library compiled from this release directory. Different versions are created for optimized and debug compile flags (OPT=O or g) and for each of the architectures, ARCH, for which the library is compiled. Thus the Opt-MS library will exist in OPTMS_DIR/lib/libOPT/ARCH/libSM.a
- src: contains the source code for the Opt-MS library. The smoothing library uses the Lapack linear solver routine, dgesv, and the necessary Lapack and BLAS routines are distributed with Opt-MS.
- src_log: contains the source code for the profiling code used in the Opt-MS library.
- util: contains shell scripts used in the configuration process

### 5.2 Configuring

The first step in installing the Opt-MS library on your a system is to create the file `Makefile.site` by using configure. This is accomplished from the OPTMS_DIR directory by typing

```
./configure [options]
```

This script will locate the C compiler, set various options, and try to locate needed libraries (such as MPI ibf parallel logging is used).

### 5.2.1 Configure Options

The following configure options are available to the user:

- **–with-arch=ARCH**: allows the user to specify the architecture for which the Opt-MS library is compiled. A utility script will try to guess this variable, but if it is unsuccessful, the configure script will halt and ask the user to reconfigure using this option.
- **–with-cc=CC**: allows the user to specify the C compiler used to compile the Opt-MS source. If no compiler is specified by the user, the configure script will attempt to find one that works on the given architecture starting with gcc.
- **–with-f77=f77**: allows the user to specify the f77 compiler used to compile the BLAS and Lapack routines distributed with Opt-MS. If no compiler is specified by the user, the configure script will attempt to find one that works on the given architecture starting with f77.
- **–with-fc_lib=FC_LIB**: allows the user to specify the f77 libraries used to link the application codes. This is needed for the Lapack and BLAS routines distributed with Opt-MS.
- **–with-opt=OPT**: allows the user to specify optimization options for the compilers (both C and Fortran), in particular 'g' and 'O' for the debugging and optimized versions, respectively. The default is O.
- **–enable-logging**: enables the profiling functionality that counts the number of times various smoothing subroutines are called and accumulates the time used in each one. The profiling code is provided in the directory `OPTMS_DIR/src_log` and will be compiled and linked with the examples provided. However, no information is accumulated for the examples unless this option is enabled during the configure process. More information on this functionality can be found in Section 4.5.
- **–enable-stats**: reports various statistics including the number of vertices smoothed, termination status information, and Laplacian and optimization-based smoothing information. More information on this functionality can be found in Section 4.4.
- **–with-debug={0,1,2,3}**: prints various levels of detail to help debug any problems with the smoothing code. Four levels of detail can be obtained:
  - Level 0 is the default and prints nothing; performance is not degraded.
  - Level 1 provides information about routines that user can access through the API and returns information such as the quality metric used for smoothing and the technique chosen.
  - Level 2 provides Level 1 information plus basic algorithmic information about smoothing of each local submesh.
  - Level 3 debugging provides the maximum amount of information including detailed information about data values and the progression of the code. Choosing the latter two options will seriously degrade performance.

  The default is level 0.
- **–enable-matlab**: for two-dimensional submeshes, enabling this option causes the smoothing code to write Matlab files that draw the initial local submesh, the search direction and new local submesh each optimization iteration, and the final local submesh. This will seriously degrade performance because of the file I/O and should be used only with Level 3 debugging on very small problems.
- **–enable-parallel**: enables the parallel option for the logging library, which allows statistics to be accumulated across the processors of a distributed memory architecture. Note that the smoothing and untangling operations are still performed locally, but global profiling and statistics information can be accumulated across processors. Enabling this functionality requires the use of the MPI standard for communication between processors.

- **–with-mpidir=MPI_DIR**: allows the user to specify the location of an installed version of MPI. The default is `/usr/local/mpi`. This is necessary only if the –enable-parallel feature has been activated.
- **–with-comm=COMM**: allows the user to specify a communication device to be used if the –enable-parallel feature is activated. The default is ch_p4, which typically supports a network of workstations. Devices supported on a given architecture are those supported by the local installation of MPI. This is necessary only if the –enable-parallel feature has been activated.

### 5.2.2 Example Configure Usage

The default options for Opt-MS configuration are

- an optimization level of O
- debugging, assertions, statistics, and profiling are turned off

Thus typing `./configure` will create a makefile that will in turn create an optimized library for which all information gathering routines such as statistics and profiling are turned off. If the user wishes to enable these options, he need only reconfigure the system with the appropriate option selected. For example, to make the optimized version of the smoothing library for the sun4 architecture with Level 1 debugging statements and with statistics and profiling enabled, one types:

```
./configure --with-arch=sun4 --enable-stats --enable-logging --with-debug=1
```

Architectures for which Opt-MS has been successfully configured and built include

- sun4 (SUN OS 4.x)
- solaris (SUN OS 5.x)

### 5.2.3 Reasons for Configure to Fail

- An architecture cannot be identified. Run configure with the option –with-arch=ARCH
- The Fortran libraries cannot be located. Run configure with the option –with-fc_lib=FC_LIB
- The C or Fortran compilers cannot be located. Run configure with the options –with-cc=CC or –with-fc=FC, respectively.
- The parallel option is enabled and the directory MPI_DIR/lib/ARCH/COMM/ does not contain a compiled MPI library. Run configure with the options –with-mpidir=MPI_DIR –with-arch=ARCH –with-comm=COMM as necessary.

## 5.3 Compiling Opt-MS

Once configure has created the file `Makefile.site`, the Opt-MS library can be made by typing

```
./make
```

This will create a directory `LIB_DIR = OPTMS_DIR/lib/libOPT/ARCH/` where OPT and ARCH are set in the configuration process. The logging library libSUMAA_log.lite.a, the Opt-MS library libSM.a, and the BLAS library, blas.a, are compiled and placed in LIB_DIR. If configure is run with the option –enable-parallel, MPI communications are used to compute the logging statistics and profiling, and the libraries libSUMAA_log.parallel_lite.a and libSM_parallel.a are created. The example codes provided with the library are also compiled, and several test cases are run.

Makefile Options

- To remake the library without remaking and running the example codes, type `./make install` which removes the current libraries from LIB_DIR, and recompiles the logging, BLAS, and Opt-MS source code.
- To remake the Opt-MS library withough remaking the logging or BLAS libraries, type `./make opt_ms`.
- To make and run the examples without remaking the libraries, type

```
./make examples; ./make runexamples
```

## 5.4 Linking Opt-MS

To compile an application that uses the Opt-MS software, the user will need to add -IOPTMS_DIR/include and -IOPTMS_DIR/log_src/include to the compile line and Opt-MS (and logging) libraries in the link line. These paths are defined in the `Makefile.site` file, which can be included in the application makefile, but the user should be cautious about overwriting Makefile variables.

As an example, a typical application makefile would contain the following lines to link the Opt-MS library into the application code.

```
TARGET = application_executable

OPTMS_DIR = /home/me/Opt-MS

OPTMS_INCLUDE = -I$(OPTMS_DIR)/include  -I$(OPTMS_DIR)/log_src/include

CFLAGS    = $(OPT)  $(OPTMS_INCLUDE)

OPTMS_LIB = -L$(OPTMS_DIR)/lib/libO/$(ARCH)  -lSM  -lSUMAAlog.lite \
                         $(OPTMS_DIR)/lib/libO/$(ARCH)/blas.a

APP_LIB = $(OPTMS_LIB) $(FORTRAN_LIB) -lm

EXAMPLESC = application_code.c
OBJSC     = application_code.o

$(TARGET): $(OBJSC) $(OPTMS_LIB)
     $(CLINKER) $(CFLAGS) -o $(TARGET) $(OBJSC) $(APP_LIB)

.c.o:
        $(CC) $(OPT) -c $(CFLAGS)  $*.c
```

Other application Makefile examples can be found in the directories OPTMS_DIR/examples/2d and OPTMS_DIR/examples/3d.

## 5.5 Running the Opt-MS Examples

A number of examples that demonstrate the features of Opt-MS are included in the directory OPTMS_DIR/examples/{2d,3d}. The main driver code in each of these directories is Smooth.c, which reads in meshes from different sources, assesses their quality, and untangles and smoothes each one.

The executables in each directory can be made from the top level directory OPTMS_DIR by typing `./make examples`, or individually by typing `./make` from the appropriate subdirectory. Once the executables are made, several default test problems can be run by typing `./make runexamples`.

In addition to the default test problems, the example codes are designed to allow the user easily to experiment with varying numbers of smoothing passes, different Opt-MS quality metrics, smoothing techniques, and threshold values. These quantities can all be changed by using command line options without recompiling the executable. The options that are available to the user are

- `-P <number of passes>`: sets the number of sweeps through the mesh
- `-M <mesh type>`: a single character that sets the input mesh type. For the examples provided, in 2D use 'T' for Triangle, 'C' for CUBIT, and in 3D use 'C' for CUBIT, 'Q' for QMG, and 'G' for GEOMPACK.
- `-i <filename>`: a character string (no longer than 128 characters) that gives the file name for the input mesh. Note that Triangle meshes are contained in two files, `meshname.ele` and `meshname.node`, and the other file types are contained in a single file, `meshname.mesh`. The input option `-i` only requires `meshname`; the appropriate suffixes are automatically appended.
- `-T <technique>`: a single character that sets the smoothing technique; use 'S' for regular Laplacian smoothing, 'L' for smart Laplacian smoothing, 'O' (capital letter o) for optimization only, 'C' for the default combined approach (approach 2), '1', '2', or '3' for the first three combined approaches described in Section 2, and 'F' for the floating threshold combined approach.
- `-F <quality metric id>`: an integer that sets the quality metric used. Each quality metric has been assigned a unique integer that is defined in the file OPTMS_DIR/include/SMuserDefs.h and described in the appendix under the subroutine pages for `SMinitSmoothing` and `SMsetSmoothFunction`.
- `-A <threshold>`: a double value that sets the threshold value for the combined approaches. Input this value in degrees for the quality metrics that are defined by element angles (such as maximize the minimum sine); for all others, input the desired threshold as it will be used in the code.

For example, to smooth the mesh contained in the files Triangle/rand400.{node,ele} using five passes of optimization-based smoothing and the "maximize the minimum sine" quality metric on a solaris machine, the user would type

```
smooth_mesh2d.solaris -M T -i Triangle/rand400 -T O -F 4 -P 5
```

The output generated by the example code includes the mesh quality information printed to stdout for each pass of untangling and smoothing, and also any statistics and profiling information that the user has enabled. In addition, a file called `meshname.qual` is generated that summarizes mesh quality information as measured by angle (dihedral angle in 3D) for each smoothing pass. Information included in this file is the minimum angle in the mesh, the maximum angle in the mesh, the average of the element minimum angles, and the angle distribution decomposed into six degree bins. Finally, in two-dimensions the smoothed mesh is printed to Triangle format (regardless of the input format) and can be viewed using the Triangle tool, `showme`. This tool can be downloaded from the URL `http://www.cs.cmu.edu/~quake/triangle.html`.

Specific 2D and 3D example meshes included with the Opt-MS package are

- 2d/Triangle: contains two meshes, `cyl200` and `rand400`, generated by the Triangle package [23]
- 2d/CUBIT: contains the `rand3200` mesh generated by the CUBIT package from Sandia National Laboratories [7]
- 2d/Tangled: contains derivatives of the meshes in the subdirectories Triangle and CUBIT that were randomly perturbed so that some percentage of the elements are invalid. There are three meshes here, `cyl200.t`, `rand400.t`, and `cubit`. These meshes are used to test the 2D untangling functionality of the Opt-MS package.
- 3d/CUBIT: contains the tetrahedral mesh `brick` generated by the CUBIT package,
- 3d/GEOMPACK: contains two of the example meshes released with GEOMPACK, `teapot` and `uobj` [16]

- **3d/QMG**: contains two of the example meshes released with QMG from Cornell, `tube` and `poly2` [24]
- **3d/Tangled**: contains tangled meshes derived from the CUBIT `brick` mesh and the QMG `tube` mesh

## 6 Getting Help

If users encounter any problems with configuring, compiling, linking, or using the Opt-MS software package, they should contact

Lori Freitag
MCS 221/C223
Argonne National Laboratory
Argonne, IL 60439
Phone: 630-252-7246
Fax: 630-252-5986
email: freitag@mcs.anl.gov

If the failure occurs in the configuration process, send a brief summary of the problem and the config.log file.

If the failure occurs with use of the library, send the local submesh that failed.

Questions and comments can also be sent to the address above, as well as suggestions for xadditional functionality.

## Acknowledgments

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

The author would like to thank the many contributions to both the theory behind the optimization-based mesh smoothing and untangling approaches, and the useability of the Opt-MS software. In particular, I thank Paul Plassmann of Pennsylvania State University, Mark Jones of Virginia Tech, Carl Ollivier-Gooch of the University of British Columbia, and Patrick Knupp of Sandia National Laboratory for their many helpful suggestions and insights.

# A Opt-MS API Appendix

SMaccumulateQualityInformation — This function computes the quality information for a right-handed triangle or tetrahedra.

## Synopsis

```
void SMaccumulateQualityInformation(void *ext_smooth_data, double **vtx)
```

## Input Parameters

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. |
| vtx | a matrix containing the coordinates of the nodes of the triangle or tetrahedra. Of dimension 3 x 2 for triangles, and 4 x 3 for tetrahedra. |

## Notes

In 2D the min, max, and average values of the triangle angles, deviation from an equilateral triangle, the scaled jacobians, and the triangle area are computed.

In 3D, the min, max, and average values of the tetrahedral dihedral angles, scaled Jacobians, the ratio of sum of the squares of the length of the edges raised to the 3/2 power to the volume, and the tetrahdral volume are computed.

## See Also

SMinitSmoothing(), SMinitQualityTable(), SMprintQualityInformation()

## Location

`SMuserFunc.c`

SMfinalizeSmoothing — This routine frees all the memory allocated in SMinitSmoothing including the data structure smooth_data. This routine should be called when mesh optimization is complete.

## Synopsis

```
void SMfinalizeSmoothing(void *ext_smooth_data)
```

## Input Parameters

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. |

## See Also

SMinitSmoothing()

## Location

`SMfree.c`

SMinitGlobalMinValue — This routine initializes the global minimum value of the quality metric to a very large value. This is used in the floating threshold technique in which the minimum value of the quality metric is tracked for the next iteration.

## Synopsis

```
void SMinitGlobalMinValue(void *ext_smooth_data)
```

## Input Parameter

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing which must be called prior to calling this routine. SMinitSmoothingStats must also have been called prior to calling this routine. |

## See Also

SMinitSmoothing(), SMsetSmoothThreshold()

## Location

`SMuserFunc.c`

SMinitQualityTable — This function allows the user to take advantage of the quality metrics implemented in the OptMS code to analyze the quality of their mesh.

## Synopsis

```
void SMinitQualityTable(void *ext_smooth_data)
```

## Input Parameter

| | |
|---|---|
| **smooth_data** | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. This routine should be called before each global pass of measuring quality or information from the previous pass will continue to be accumulated. |

## Note

In 2D the min, max, and average values of the triangle angles, deviation from an equilateral triangle, the scaled jacobians, and the triangle area are printed.

In 3D, the min, max, and average values of the tetrahedral dihedral angles, scaled Jacobians, the ratio of sum of the squares of the length of the edges raised to the 3/2 power to the volume, and the tetrahdral volume are printed.

If any triangle areas or tetrahedral volumes are negative, the mesh is considered to be invalid, and SMuntangle should be called to try to create a valid mesh.

## See Also

SMinitSmoothing(), SMuntangle(), SMaccumulateQualityInformation(), SMprintQualityInformation()

## Location

`SMuserFunc.c`

SMinitSmoothStats — If statistics gathering has been enabled in the configure process, then this routine intializes the statistics structure that records the number of cells smoothed, the number of equilibrium points found, and the reason for algorithm termination.

## Synopsis

```
void SMinitSmoothStats(void *ext_smooth_data)
```

## Input Parameter

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. |

## Note

It is useful to call this routine in conjunction with SMprintSmoothStats during each global pass over the mesh so that the incremental improvment can be monitored.

## See Also

SMprintSmoothStats()

## Location

`SMuserFunc.c`

**SMinitSmoothing** — Initializes the smoothing data structure and sets values for the smoothing technique, the mesh quality function, and threshold usage.

## Synopsis

```
void SMinitSmoothing(int argc, char** argv, int dimension,
                     char technique, int FunctionID, double Threshold,
                     void **ext_smooth_data)
```

## Input Parameters

| | |
|---|---|
| argc | the number of input arguments to the program; used to initialize logging. The logging library checkes for the command line argument -logfile filename to print the log summary to a file rather then to stdout. If logging is not desired, or argc and argv are not available, NULL may be passed in for these arguements. |
| argv | input arguments to the program; used to initialize logging |
| dimension | an integer indicating the problem dimension, either 2 or 3 |
| technique | a character argument to set the smoothing technique used to adjust grid point location. |

```
        Input one of:
               OPTMS_LAPLACIAN_ONLY       (or L)
               OPTMS_OPTIMIZATION_ONLY    (or O)
               OPTMS_COMBINED             (or C)
               OPTMS_COMBINED1            (or 1)
               OPTMS_COMBINED2            (or 2)
               OPTMS_COMBINED3            (or 3)
               OPTMS_FLOATING_THRESHOLD   (or F)
               OPTMS_STUPID_LAPLACE       (or S)
               OPTMS_TECHNIQUE_DEFAULT    (or C)
               OPTMS_DEFAULT              (or -1)

          Note that either the COMBINED or FLOATING_THRESHOLD technique
          is recommended.  The COMBINED approach is the default.
```

| | |
|---|---|
| FunctionID | an integer argument used to set the mesh quality measure to be optimized. |

```
       In 2D input one of
            MAX_MIN_ANGLE  (or 1): maximize the minimum angle
            MIN_MAX_COSINE (or 2): minimize the maximum cosine of the angle
            MAX_MIN_COSINE (or 3): maximize the minimum cosine of the angle
            MAX_MIN_SINE   (or 4): maximize the minimum sine of the angle
            MIN_MAX_ANGLE  (or 5): minimize the maximum angle
            MIN_MAX_JACOBIAN_DIFF (or 6): minimize the maximum square of the
                difference of the current jacobian and the jacobian of an equilateral
                triangle (scaled by the jacobian of an equilateral triangle)
            MAX_MIN_SCALED_JACOBIAN (or 7): maximize the minimum scaled jacobian
                  for each of the three vertices of a triangle (J/(L1*L2)) where L1 and L2 are
                  the lengths of the incident edges.  Same as MAX_MIN_SINE in the feasible
                  region, but returns negative angles for inverted elements
```

```
      MAX_MIN_AREA_LENGTH_RATIO (or 8):  Computes the ratio of the the area
            of the triangle and the sum of the squares of the length of the edges
      MIN_MAX_LENGTH_AREA_RATIO (or 9): Computes the negtive inverse of the
            MAX_MIN_AREA_LENGTH_RATIO
      FUNCTION_DEFAULT_2D (which is MAX_MIN_SINE) (or 4)
      OPTMS_DEFAULT (-1, which will result in a choice of MAX_MIN_SINE)

    In 3D input one of
       MAX_MIN_DIHEDRAL (or 21): maximize the minimum angle
       MIN_MAX_DIHEDRAL (or 22): minimize the maximum angle
       MAX_MIN_COSINE_DIHEDRAL (or 23): maximize the minimum cosine of the angle
       MIN_MAX_COSINE_DIHEDRAL (or 24): minimize the maximum cosine of the angle
       MAX_SINE_DIHEDRAL (or 25): maximize the minimum sine of the angle
       FUNCTION_DEFAULT_3D (which is MAX_SINE_DIHEDRAL) (or 25)
       OPTMS_DEFAULT (-1, which will result in a choice of MAX_SINE_DIHEDRAL)

The default in 2D is MAX_MIN_SINE and in 3D is MAX_SINE_DIHEDRAL.
```

| | |
|---|---|
| Threshold | a double argument that sets the degree value of the threshold used in either the COMBINED technique, which has a fixed value, or the FLOATING_THRESHOLD technique, which allows the threshold to vary. The default values for the quality measures that depend on angle measures are 30 and 15 degrees for the COMBINED approach for 2D and 3D, respectively, and 10 and 15 degrees for the FLOATING_THRESHOLD approach. For the measures that depend on Jacobian ratios the default is .25. |

## Output Parameters

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing |

## See Also

SMsetSmoothTechnique(), SMsetSmoothFunction(), SMsetSmoothThreshold(), SMfinalizeSmoothing()

## Location

SMuserFunc.c

SMinvalidMesh — This function returns a Boolean value after testing to see if the mesh contains any invalid elements. The test is based on information contained in the quality table and this information must be accumulated before this test can be made. This is useful in determining if the mesh requires untangling.

## Synopsis

```
int SMinvalidMesh(void *ext_smooth_data)
```

## Input Parameters

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. |

## Return Value

a Boolean value of 1 if the mesh contains invalid elements or 0 if the mesh contains no invalid elements.

## See Also

SMinitSmoothing, SMinitQualityInformation(), SMaccumulateQualityInformtaion,
SMuntangle()

## Location

```
SMuserFunc.c
```

**SMprintQualityInformation** — This function prints the quality information accumulated since the last call to SMinitQualityInformation.

## Synopsis

```
void SMprintQualityInformation(void *ext_smooth_data)
```

## Input Parameters

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. |

## Notes

In 2D the min, max, and average values of the triangle angles, deviation from an equilateral triangle, the scaled jacobians, and the triangle area are printed.

In 3D, the min, max, and average values of the tetrahedral dihedral angles, scaled jacobians, the ratio of sum of the squares of the length of the edges raised to the 3/2 power to the volume, and the tetrahdral volume are printed.

If any triangle areas or tetrahedral volumes are negative, the mesh is considered to be invalid, and SMuntangle should be called to try to create a valid mesh.

## See Also

SMinitSmoothing(), SMinitQualityInformation(), SMaccumulateQualityInformation()

## Location

`SMuserFunc.c`

SMprintSmoothStats — If statistics gathering has been enabled in the configure process, then this routine prints the statistics that have been accumulated by the smoothing code since the last call to SMinitSmoothStats.

## Synopsis

```
void  SMprintSmoothStats(void *ext_smooth_data)
```

## Input Parameter

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing which must be called prior to calling this routine. SMinitSmoothingStats must also have been called prior to calling this routine. |

## Note

```
   The information printed includes
        - the total number of nodes smoothed,
        - the number for which Laplacian smoothing was used and the number
          of the those that resulted in an invalid mesh and/or no improvement to
          the mesh
        - the number of nodes for which optimization-based smoothing was used
          (including the average iteration count),
        - the number of cells with no improvement,
        - the averate active value and average improvement, and
        - the termination status for the cells that were smoothed.
```

## See Also

SMinitSmoothing(), SMinitSmoothStats()

## Location

`SMuserFunc.c`

SMsetMeshValidity — This function allows the user to set the mesh validity without using the quality functions provided by the Opt-MS package. This is useful if the user knows the mesh requires untangling and doesn't want to evaluate the quality of the entire mesh to determine if there are inverted elements.

## Synopsis

```
void SMsetMeshValidity(int mesh_validity, void *ext_smooth_data)
```

## Input Parameters

| | |
|---|---|
| mesh_validity | a Boolean value of 1 if the mesh is valid and 0 if the mesh contains elements with negative area |
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. |

## See Also

SMinitSmoothing, SMinitQualityInformation(), SMaccumulateQualityInformtaion, SMuntangle()

## Location

`SMuserFunc.c`

SMsetProblemDimension — This function allows the user to set the dimension of the smoothing problem. Opt-MS currently supports 2D planar smoothing (in the x-y plane) and 3D smoothing. This function call be called any time after SMinitSmoothing to set or change the dimension of the problem.

## Synopsis

```
void SMsetProblemDimension(void *smooth_data, int dimension)
```

## Input Parameters

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. |
| dimension | an integer argument to set the dimension of the problem, either 2 or 3. |

## See Also

SMinitSmoothing()

## Location

`SMuserFunc.c`

**SMsetSmoothFunction** — This function allows the user to change the mesh quality function that is optimized at any time during the smoothing process.

## Synopsis

```
void SMsetSmoothFunction(void *ext_smooth_data,int FunctionID)
```

## Input Parameters

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. |
| FunctionID | an integer argument used to set the mesh quality measure to be optimized. |

```
In 2D input one of
      MAX_MIN_ANGLE  (or 1): maximize the minimum angle
      MIN_MAX_COSINE (or 2): minimize the maximum cosine of the angle
      MAX_MIN_COSINE (or 3): maximize the minimum cosine of the angle
      MAX_MIN_SINE   (or 4): maximize the minimum sine of the angle
      MIN_MAX_ANGLE  (or 5): minimize the maximum angle
      MIN_MAX_JACOBIAN_DIFF (or 6): minimize the maximum square of the
          difference of the current jacobian and the jacobian of an equilateral
          triangle (scaled by the jacobian of an equilateral triangle)
      MAX_MIN_SCALED_JACOBIAN (or 7): maximize the minimum scaled jacobian
            for each of the three vertices of a triangle (J/(L1*L2)) where L1 and L2 are
            the lengths of the incident edges.  Same as MAX_MIN_SINE in the feasible
            region, but returns negative angles for inverted elements
      MAX_MIN_AREA_LENGTH_RATIO (or 8):  Computes the ratio of the the area
            of the triangle and the sum of the squares of the length of the edges
      MIN_MAX_LENGTH_AREA_RATIO (or 9): Computes the negtive inverse of the
            MAX_MIN_AREA_LENGTH_RATIO
      FUNCTION_DEFAULT_2D (which is MAX_MIN_SINE) (or 4)
      OPTMS_DEFAULT (-1, which will result in a choice of MAX_MIN_SINE)

     In 3D input one of
       MAX_MIN_DIHEDRAL (or 21): maximize the minimum angle
       MIN_MAX_DIHEDRAL (or 22): minimize the maximum angle
       MAX_MIN_COSINE_DIHEDRAL (or 23): maximize the minimum cosine of the angle
       MIN_MAX_COSINE_DIHEDRAL (or 24): minimize the maximum cosine of the angle
       MAX_SINE_DIHEDRAL (or 25): maximize the minimum sine of the angle
       FUNCTION_DEFAULT_3D (which is MAX_SINE_DIHEDRAL) (or 25)
       OPTMS_DEFAULT (-1, which will result in a choice of MAX_SINE_DIHEDRAL)

  The default in 2D is MAX_MIN_SINE and in 3D is MAX_SINE_DIHEDRAL.
```

## See Also

SMinitSmoothing()

### Location

`SMuserFunc.c`

SMsetSmoothTechnique — This function allows the user to change the technique used for mesh smoothing at any time during the mesh improvement process.

## Synopsis

```
void SMsetSmoothTechnique(void *ext_smooth_data, char technique)
```

## Input Parameters

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. |
| technique | a character argument to set the smoothing technique used to adjust grid point location. |

```
         Input one of:
                OPTMS_LAPLACIAN            (or S)
                OPTMS_SMART_LAPLACIAN      (or L)
                OPTMS_OPTIMIZATION_ONLY    (or O)
                OPTMS_COMBINED             (or C)
                OPTMS_COMBINED1            (or 1)
                OPTMS_COMBINED2            (or 2)
                OPTMS_COMBINED3            (or 3)
                OPTMS_FLOATING_THRESHOLD   (or F)
                OPTMS_STUPID_LAPLACE       (or S)
                OPTMS_TECHNIQUE_DEFAULT    (or C)
                OPTMS_DEFAULT              (or -1)

           Note that either the COMBINED or FLOATING_THRESHOLD technique
           is recommended.  The COMBINED approach is the default.
```

```
      Note that either the COMBINED or FLOATING_THRESHOLD technique
      is recommended.  The COMBINED approach is the default.
```

## See Also

SMinitSmoothing()

## Location

```
SMuserFunc.c
```

SMsetSmoothThreshold — This function allows the user to change the value of the threshold when using the combined or the FLOATING_THRESHOLD techniques at any time in the smoothing process.

## Synopsis

```
void SMsetSmoothThreshold(void *ext_smooth_data, double Threshold)
```

## Input Parameters

| | |
|---|---|
| smooth_data | a void data structure that contains the context and data structures for smoothing. This structure is created in SMinitSmoothing, which must be called prior to calling this routine. |
| Threshold | a double argument that sets the degree value of the threshold used in either the COMBINED technique, which has a fixed value, or the FLOATING_THRESHOLD technique, which allows the threshold to vary. The default values for the quality measures that depend on angles are 15 and 30 degrees for the COMBINED approach for 3D and 2D respectively, and 10 and 15 degrees for the FLOATING_THRESHOLD approach. For the measures that depend on Jacobian ratios the default is .25. |

## See Also

SMinitSmoothing()

## Location

```
SMuserFunc.c
```

SMsmooth — This is the main routine that optimizes a local submesh. Most of the quality functions available require that the local submesh is initially valid. If the user suspects that the mesh contains invalid elements, a quality assessment should be done, and, if necessary, SMuntangle should be used to try to rectify the problem.

## Synopsis

```
int SMsmooth(int num_incident_vtx, int num_tri, double *free_vtx,
             double **vtx_list, int **vtx_connectivity,
             void *ext_smooth_data)
```

## Input Parameters

| | |
|---|---|
| num_incident_vtx | the number of incident vertices in the local mesh |
| num_tri | the number of incident triangles or tetrahedra |
| free_vtx | the coordinates of the free vertex a vector of length equal to the problem dimension in x, y, z order |
| vtx_list | a matrix of the coordinates of the incident vtx; matrix dimensions are num_incident_vtx by problem dimension |
| vtx_connectivity | a matrix that gives the connectivity info for the incident vertices. matrix dimensions are num_incident_vtx by the problem dimension. Note: this assumes that the connectivity given is for a right handed triangle or tetrahedra the free vertex ordered first |
| ext_smooth_data | data structure for mesh smoothing; created in SMinitSmoothing and cast to the local data structure |

## Output Parameter

| | |
|---|---|
| free_vtx | contains the new coordinates of the free vertex |

## Note

This function can be called only after SMinitSmoothing has been called to create the ext_smooth_data data structure. Once the mesh has been optimized, SMfinalizeSmoothing should be called to release the memory.

## See Also

SMinitSmoothing(), SMsetSmoothTechnique(), SMsetSmoothFunction(), SMsetSmoothThreshold(), SMfinializeSmoothing()

## Location

`SMsmooth.c`

SMuntangle — This is the main routine that attempts to untangle a local submesh by maximizing the minimum area of a triangle or tetrahedra in a local submesh. The value of the function assumes a right-hand ordering of the vertices. If the user is not sure if the mesh is invalid, a quality assessment should be done to compute the Jacobians of each of the mesh elements. Once this information is accumulated, the routine SMinvalidMesh returns TRUE (1) if the mesh contains inverted elements.

## Synopsis

```
int SMuntangle(int num_incident_vtx, int num_tri, double *free_vtx,
              double **vtx_list, int **vtx_connectivity,
              void *ext_smooth_data)
```

## Input Parameters

| | |
|---|---|
| num_incident_vtx | the number of incident vertices in the local mesh |
| num_tri | the number of incident triangles or tetrahedra |
| free_vtx | the coordinates of the free vertex in a vector of length problem dimensions in x, y, z order |
| vtx_list | a matrix of the coordinates of the incident vtx; matrix dimensions are num_incident_vtx by problem dimension |
| vtx_connectivity | a matrix that gives the connectivity info for the incident vertices. matrix dimensions are num_incident_vtx by the problem dimension. Note: this assumes that the connectivity given is for a right handed triangle or tetrahedra with the free vertex ordered first |
| ext_smooth_data | data structure for mesh smoothing; created in SMinitSmoothing and cast to the local data structure |

## Output Parameter

| | |
|---|---|
| free_vtx | contains the new coordinates of the free vertex |

## Note

This function can only be called after SMinitSmoothing has been called to create the smooth_data data structure. Once the mesh has been untangled, SMsmooth should be called to improve the element quality as it is usually very poor at the conclusion of this step.

## See Also

SMinitSmoothing(), SMinitQualityTable, SMinvalidMesh(), SMfinializeSmoothing()

## Location

`SMuntangle.c`

## References

[1] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. In *8th ACM-SIAM Symp. on Discrete Algorithms*, pages 528–537, 1997.

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

[3] R. E. Bank and R. K. Smith. Mesh smoothing using a posteriori error estimates. *SIAM Journal on Numerical Analysis*, 34(3):979–997, June 1997.

[4] Scott Canann, Michael Stephenson, and Ted Blacker. Optismoothing: An optimization-driven approach to mesh smoothing. *Finite Elements in Analysis and Design*, 13:185–190, 1993.

[5] C. Charalambous and A. Conn. An efficient method to solve the minimax problem directly. *SIAM Journal of Numerical Analysis*, 15(1):162–187, 1978.

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

[7] T. D. Blacker et al. CUBIT mesh generation environment. Technical Report SAND94-1100, Sandia National Laboratory, Albuquerque, New Mexico, May 1994.

[8] David A. Field. Laplacian smoothing and Delaunay triangulations. *Communications and Applied Numerical Methods*, 4:709–712, 1988.

[9] Lori Freitag. On combining Laplacian and optimization-based smoothing techniques. In *Trends in Unstructured Mesh Generation*, volume AMD-Vol. 220, pages 37–44. ASME Applied Mechanics Division, 1997.

[10] Lori Freitag and Carl Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal of Numerical Methods in Engineering*, 40:3979–4002, 1997.

[11] Lori Freitag and Carl Ollivier-Gooch. A cost/benefit analysis of simplicial mesh improvement as measured by solution efficiency. Preprint ANL/MCS-P722-0598, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1998. also to appear in International Journal of Computational Geometry.

[12] Lori Freitag and Paul Plassmann. Local optimization-based simplicial mesh untangling and improvement. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory.

[13] Lori A. Freitag, Mark T. Jones, and Paul E. Plassmann. An efficient parallel algorithm for mesh smoothing. In *Proceedings of the Fourth International Meshing Roundtable*, pages 47–58. Sandia National Laboratories, 1995.

[14] P. Gill, W. Murray, and Margaret Wright. *Practical Optimization.* Academic Press, 1981.

[15] Barry Joe. Three-dimensional triangulations from local transformations. *SIAM Journal on Scientific and Statistical Computing*, 10:718–741, 1989.

[16] Barry Joe. Geompack - a software package for the generation of meshes using geometric algorithms. *Advanced Engineering Software*, 56(13):325–331, 1991.

[17] Barry Joe. Construction of three-dimensional improved quality triangulations using local transformations. *SIAM Journal on Scientific Computing*, 16:1292–1307, 1995.

[18] Patrick Knupp. Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities, Part 1 - A framework for surface mesh optimization. Technical Report SAND 99-0110J, Sandia National Laboratory, 1999.

[19] S. H. Lo. A new mesh generation scheme for arbitrary planar domains. *International Journal for Numerical Methods in Engineering*, 21:1403–1426, 1985.

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

[21] Matthew L. Staten Scott A. Canann, Joseph R. Tristano. An approach to combined Laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes. In *Proceedings of the 7th International Meshing Roundtable*, pages 479–494. Sandia National Laboratory, 1998.

[22] Mark Shephard and Marcel Georges. Automatic three-dimensional mesh generation by the finite octree technique. Technical Report SCOREC Report No. 1-1991, Scientific Computation Research Center, Rensselaer Polytechnic Institute, 1991.

[23] Jonathan Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Proceedings of the First Workshop on Applied Computational Geometry*, pages 124–133, Philadelphia, Pennsylvania, May 1996. ACM.

[24] Steve Vavasis and Scott Mitchell. Quality mesh generation in higher dimensions. Technical Report ANL/MCS-P633-1296, Argonne National Laboratory, 1996.