Sandia National Laboratories

Operated for the U.S. Department of Energy by
Sandia Corporation

Albuquerque, New Mexico 87185- 0834

date: September 15, 2003
Original September 15, 2003

to: Distribution

from: E. D. Wilkes, MS-0834 (GRAM, Inc.)

subject: Using Element Quality Metrics in GOMA (GT-025.0)

keywords: finite elements, element quality, meshing, remeshing, scripts

input records: none

Introduction

In many moving boundary application problems, the volume of a material may vary in time if it has a net inflow/outflow or if it participates in a chemical reaction or a process such as evaporation or deposition. These phenomena may significantly change the shape and/or size of the material's domain over time. This requires one or more of the surfaces in a finite element mesh to move along with the physical boundary it represents, which in turn requires the geometry of finite elements in regions adjacent to such surfaces to conform to the evolving domain shape. As these elements undergo shape changes, they may develop undesirable features such as excessive aspect ratios, very large or very small vertex angles, or even overturning. Element quality may be thought of as the absence of such detrimental attributes, in that they increase the difficulty in proceeding to the next time step, and may also increase the error in the computed solution. That is, the success of a finite algorithm in solving moving-boundary problems is dependent on maintaining acceptable element quality.

Maintaining element quality often requires the program to be stopped and restarted with a new mesh in order to continue pursuing a transient solution; this is referred to as a remeshing/ remapping step. The question which then arises is, "How does the program determine when a remeshing/remapping step is necessary"? If these steps are not done often enough, then poor element quality will increase the error introduced by interpolating the solution onto the new mesh. On the other hand, doing the remeshing/remapping steps more often than necessary is safer, but increases the computational overhead/expense for the problem. Therefore, what is desired is to be able to monitor the degree of element distortion and use this information to make intelligent decisions about remeshing intervals. Hence, *Goma* has been provided with four element quality metrics which can be computed at each time step, and a means for setting a criterion for the next remesh/remap step.

The following element quality metrics are available for 2D quadrilateral elements:

- Jacobian: Ratio of the smallest elemental Jacobian value at the vertices to the average value at the vertices [1].

- Volume change: Uses the volume change component of mesh stress (collected during problem assembly).
- Angle: Measures L2 norm of the deviation of vertex angles from $\pi/2$. The metric is then 1 - |L2|/($\pi/2$).
- Triangle: Computes Lo's [1,2]quality metric for each of two pairs of subtriangles formed by dividing the quadrilateral along each of its diagonals. These four metric values are then sorted such that $q1 < q2 < q3 < q4$, and the overall metric is q1q2/q3q4.

These metrics are presently limited to 2D meshes. Note that each of the metrics is computed such that a value of 1 is "ideal" and a value of 0 is "unacceptable". Thus, the higher the metric value, the better the element quality. The volume change metric, however, can exceed 1 when the elements increase in size normally.

Invoking and using element quality metrics

The element quality metrics may be invoked for almost any 2D free surface problem that *Goma* can solve. When one or more metrics are selected, a quality table is generated at the beginning of the solution and after each step. This table provides the average and minimum values of each chosen metric among the elements. If more than one metric is active, the table will also include a combined weighted average and global minimum among the active metrics. The default is not to do any of these metrics, so each desired metric must be specifically requested by including one or more of the following cards at the end of the "`Time Integration Specifications`" section of the *Goma* input file:

```
Jacobian quality weight = 1.0

Volume change quality weight = 1.0

Angle quality weight = 1.0

Triangle quality weight = 1.0
```

Note that the float input for each of the cards is a weighting factor. These floats may be used to generate a custom composite metric in which some metrics may be weighted more heavily than others for the purpose of computing the combined metric average. In this case, each may be given different positive values (which do not have to add up to 1.0). Otherwise, these float should normally be left at 1.0.

The primary purpose of using element quality metrics is to terminate a run before a point is reached where the elements become so distorted as to cause abnormal program termination or significant accuracy loss in the solution. When there are one or more active metrics, a tolerance value between 0.0 and 1.0 may be specified. There are two additional input cards associated with this tolerance:

```
Element quality tolerance = <float>

Element quality tolerance type = {avg | min | jac | vol | ang | tri}
```

The first card inputs the tolerance value (the default is 0). The second card allows a selection of the value which is to be compared to the specified tolerance to determine whether or not to proceed with computations. `avg` indicates the weighted average of the metrics. `min` indicates the minimum of all active metrics (this is the default). The other choices allow one specific metric to be used for the stopping criterion, even when other metrics are active.

Tutorial problems

This tutorial comes with two example problems which demonstrate the use of *Goma* with element quality metrics to monitor the degree of element distortion during computations and terminate a run when distortion becomes excessive. The ability to solve a problem over a long time period with remesh/remap steps taken only when necessary will also be demonstrated.

The basic requirements for running these tutorials are a current *Goma* executable which includes Umfpack and Aztec, APREPRO, plus CUBIT and MAPVAR for remeshing and remapping. One problem involves multiple remesh/remap steps and has been provided with automatic scripts to manage all tasks; these scripts require the `sed` utility and GROPE (included with SEACAS) to create input decks for each step. There are two versions of each script (serial and parallel). Running in parallel additionally requires a parallel-built *Goma* executable and the `brk` and `fix` utilities. The parallel scripts are set up to run *Goma* on 8 processors; if running on a system with fewer available processors (or if a different number is desired), edit the parallel script version by replacing 8 with the desired number of processors.

Liquid spill from a heated container

In this problem, a fluid (density = 1 g/cc, viscosity = 10 cP) is initially at rest inside a rectangular container of width 5.08 cm filled to a height of 6.35 cm with a flat meniscus. The bottom of the container is maintained at 90 K while a steady heat flux of 4.2 W/cm^2 enters through the top. The container is turned over on one side at time zero, and the fluid begins to flow outward.

The files provided with the tutorial distribution (subdirectory “sideheat”) are as follows:

- generic.mat - fluid property inputs
- Defs_noscale - APREPRO file with problem dimension data
- sidesharp.jou - CUBIT input file for initial mesh
- input.noeq - Goma input file without element quality metrics
- input.eq - Goma input file with element quality metrics
- cleanup - Script file which cleans the directory after a run

Procedures

To run the problem, insert the *Goma* executable (or a link to it) in the problem directory and ensure that the required utilities are accessible from this directory. Generate the ExodusII input file “sidesharp.exoII” with the command:

```
cubit sidesharp.jou
```

To illustrate the effectiveness of the element quality metrics, first run the case without using them:

```
goma -a -i input.noeq
```

This will require about 200 MB of memory. After 43 steps, one convergence failure will occur. When the step is retried at half the time step size, one more converged solution will be obtained, but by this time the unmonitored element distortion has become excessive, as indicated in Figure 1:

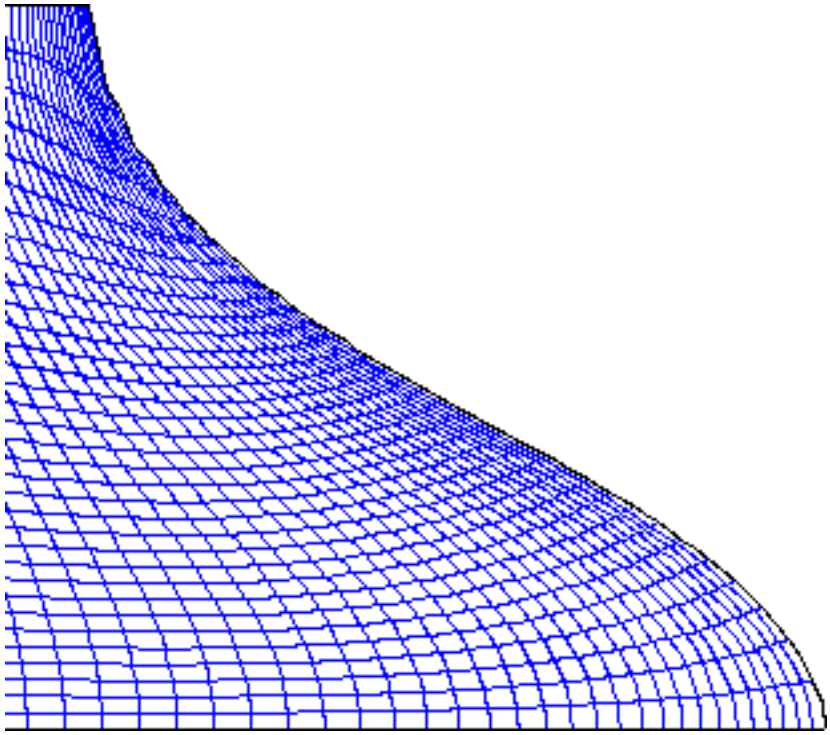

Figure 1. Distorted free surface discretization near finite element failure.

Figure 1 shows that as the free surface deforms, the shapes of the interior elements undergo marked distortion while trying to conform to the domain shape. Another consequence of this is that the length of the meniscus increases to the point where the number of nodes on the free surface is no longer sufficient to accurately represent it. These problems display the symptoms of increasing difficulty in obtaining convergence and eventually failure of the linear solver as the Jacobian matrix becomes singular.

This is where the element quality metrics come in. Now add the four metrics to the problem specifications with a criterion (tolerance) of 0.3 on the global minimum, and re-run the problem:

```
goma -a -i input.eq
```

This runs the same problem, only with the metrics. Before the first time step begins, an initial element quality check is done, producing the following table:

```
INITIAL ELEMENT QUALITY CHECK---
 ELEMENT QUALITY METRIC         AVG              MIN
               Jacobian         0.694444         0.694444
               Angle                   1                1
               Triangle                1                1
               COMBINED         0.898148         0.694444
Element quality OK!
```

Note that the volume change metric was not done even though the card was included in the input deck; this is because the data for this metric are collected during assembly of the mesh equations, which has not been done yet. After the first step converges, all four metrics are recomputed and the following table is generated:

```
ELEMENT QUALITY METRIC
               Jacobian         0.694444         0.694444
               Volume change           1                1
               Angle                   1                1
               Triangle                1                1
               COMBINED         0.923611         0.694444
Element quality OK!
```

The calculations then proceed as before, except that this table is displayed after each step. A total of 42 steps are completed, after which the table shows:

```
 ELEMENT QUALITY METRIC         AVG              MIN
               Jacobian         0.684263         0.484691
               Volume change     1.01626         0.480432
               Angle            0.874674         0.733659
               Triangle         0.585668         0.283246
               COMBINED         0.790217         0.283246
Element quality below tolerance of 0.3
        REMESHING IS REQUIRED!
```

At this point, the triangle metric has fallen below the specified tolerance of 0.3 for element 665, signaling the end of the *Goma* run. This is indicated by the display line “REMESHING IS REQUIRED”.

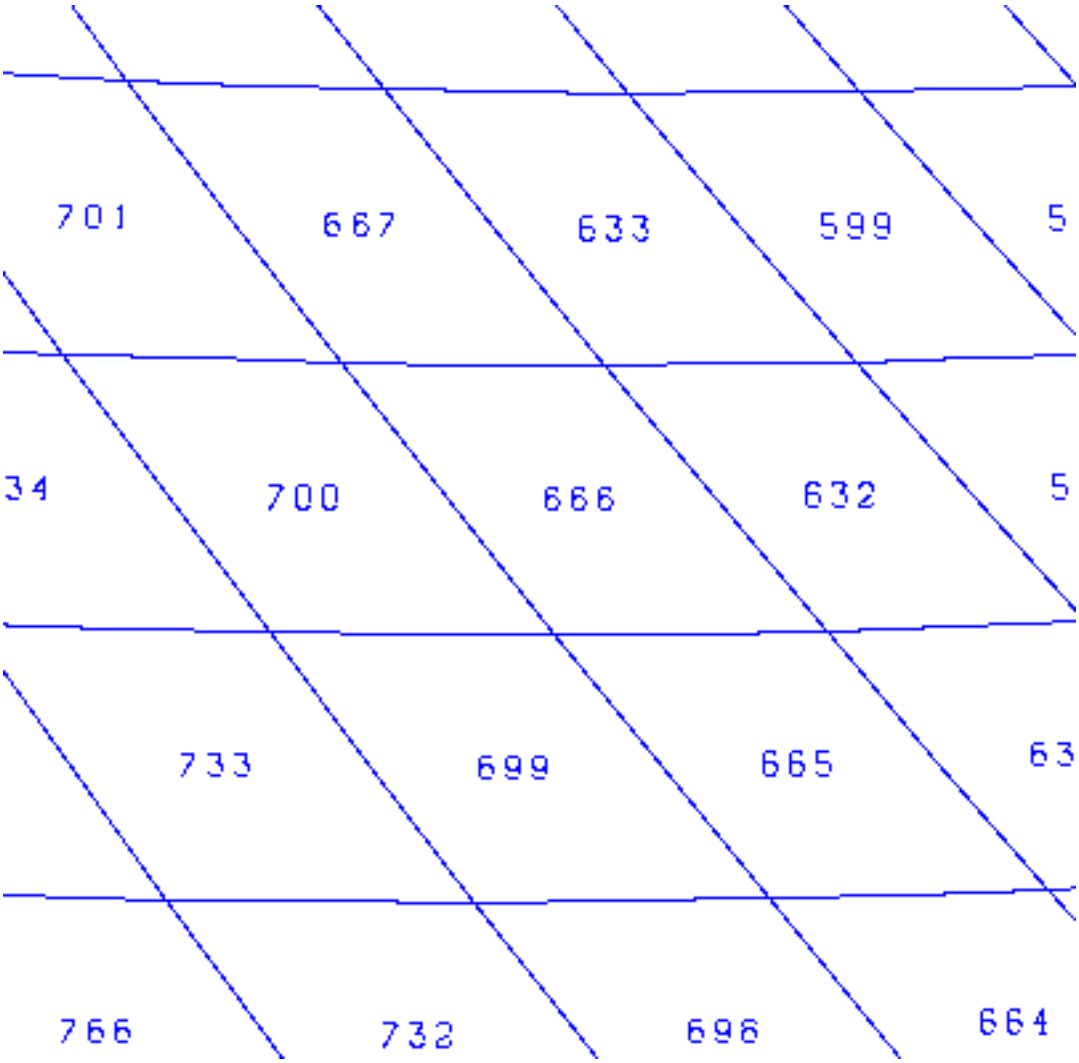

Figure 2. Distorted mesh at final time step: Element 665 triangle metric is 0.283.

In this case, as shown by Figure 2, one of the metrics detected significant element distortion before either of the problems noted in the first run appeared. This makes it possible to avoid attempts to advance a solution which is starting to be adversely affected by poor element quality -- even if more steps can be taken, the computational effort necessary to continue time integration will increase and the element quality and solution accuracy will suffer even more. If desired, it is possible to create a new mesh and interpolate the final solution onto it so that *Goma* could be restarted to carry the computations farther in time. The next problem illustrates this ability.

Copper sulfidation and layer formation from a pinhole

This problem examines the growth of a copper sulfide layer on an electrical contact. The contact is made of metallic copper with a thin layer of gold electroplating which protects the copper from exposure to the atmosphere. However, such thin layers are susceptible to very small "pinhole" defects through which exposure occurs. Copper has been shown [3,4,5] to react with hydrogen sulfide in the

ambient air, even at minute (ppb scale) concentrations, to form a layer of copper sulfide in the vicinity of the defect. This layer allows Cu^{-} ions to diffuse through to the surface at a rate which is sufficient to maintain the sulfidation reaction. The sulfide layer then continues to expand outward and cover a larger fraction of the contact surface over time. This increases the electrical resistance of the surface, leading to contact failure.

The copper sulfidation reaction mechanism and kinetics have been extensively studied [4,5] and are relatively straightforward to model. The real challenge is to maintain a satisfactory mesh quality while the sulfide domain grows from a thin layer to a large mass relative to the pinhole. This is handled by breaking the solution into several sequential runs, with a new mesh created for each [6]. Such a solution is typically managed by a shell script, from which commands are issued for each step of the process. In doing so, the question of how often to perform the remesh/remap procedure must be addressed. An interval may be set based on a given time increment or number of steps, but this does not ensure that any given run will be neither too short or too long. This example problem illustrates how element quality metrics can be used to determine the optimal meshing intervals based on the actual degree of element distortion.

The files provided with the tutorial distribution (subdirectory “pinhole”) are as follows:

- sulfide.mat - sulfide layer property inputs
- ec.cubit - CUBIT input file for creating initial mesh
- ec-remesh.cubit - CUBIT input file for remeshing after each run
- input.startup - *Goma* input file for first run
- input.template - Base input file from which custom input files are created
- mapvar_input - Instruction file for MAPVAR when called from a script
- grope.input - Instruction file for GROPE utility when called from a script
- ec.serial.sh - Unix shell script for serial computation
- ec.parallel.sh - Unix shell script for parallel computation
- brk_yd.in - Input file for brk utility
- cleanup - script file to clean unneeded files from directory when done

This problem will require about 690 MB of memory for a serial run.

Procedures

It is instructive to first carry out each of the steps manually, then with a script. Start by generating the initial mesh in CUBIT:

```
cubit ec.cubit
```

CUBIT will open a window in which the new mesh is displayed; to suppress this window, add -nographics to the command line arguments. CUBIT creates a file called "ec.exoII" which contains all of the information for the new mesh which was just generated. Now, copy the given startup input file for *Goma*:

```
cp input.startup input
```

Ensure that the *Goma* executable can be accessed from the working directory, then perform the first *Goma* run:

```
goma
```

This run will take 39 time steps with a final step from 291222 s to 436833 s (step size 145611), stopping when the triangle metric falls below 0.15:

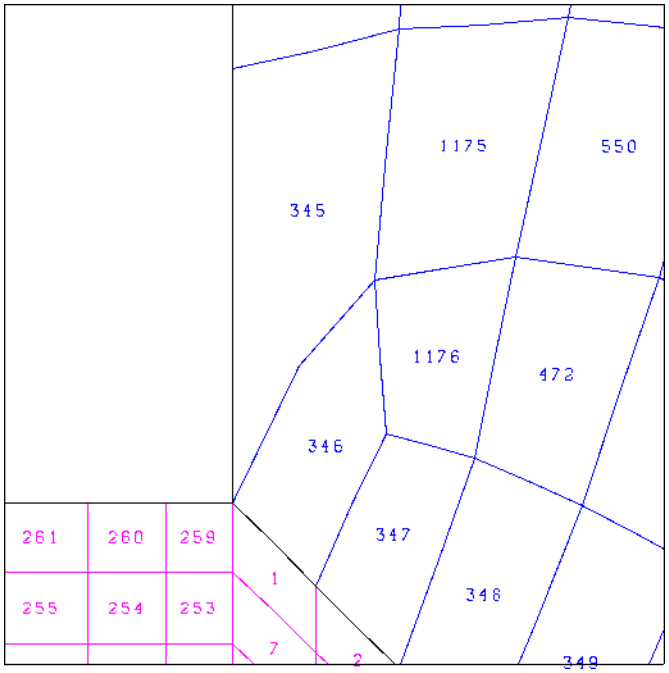

Figure 3. Distorted mesh at final time step. Element 345 triangle metric is 0.118.

At this point (shown in Figure 3), the quality output table is as follows:

```
 ELEMENT QUALITY METRIC          AVG              MIN
               Jacobian          0.671465         0.418121
               Volume change      3.69615          3.27081
               Angle             0.918051         0.685873
               Triangle          0.704622         0.118004
               COMBINED           1.49757         0.118004
Element quality below tolerance of 0.15
        REMESHING IS REQUIRED!
```

Figure 3 shows that the extent of element distortion clearly calls for a remesh, and the triangle metric relayed this information to the transient solver in *Goma* to stop the run, as it was intended to do.

Before proceeding, save the input and output files as desired for the initial run:

```
cp ec.exoII ec.exoII.t0
cp ec.out.exoII ec.out.exoII.t0
cp input input.t0
```

From this point, the computation can be continued with a series of remesh/remap/rerun steps until some desired criterion (e.g. total time) is met. Instructions for step 1 (to perform the second run) are provided here; for subsequent steps just replace 1 with the step number.

--- START OF LOOP ---

Use the provided remesh journal file to generate the next mesh in CUBIT:

```
cubit ec-remesh.cubit
```

This creates a new mesh file called "ec-new.exoII". The next step is to interpolate the last steady state solution from the old mesh onto the new mesh. MAPVAR performs this function, but also requires its own copies of the old output file and the new mesh file:

```
cp ec-new.exoII ec.g
cp ec.out.exoII ec.e
```

Mapvar is then invoked as follows:

```
mapvar ec
```

This tells MAPVAR to transfer the last solution in ec.e to the mesh in ec.g and write the new mesh with the interpolated solution to a new file "ec.int". After MAPVAR is started, it will prompt for a menu choice. Type:

```
def 2
```

This will tell MAPVAR to reset the mesh node coordinates such that the displacements will start at zero when *Goma* is restarted. This is referred to as "annealing" in *Goma*. At the next prompt, type:

```
run
```

When MAPVAR stops, the new mesh file "ec.int" will contain the new mesh and the previous solution interpolated onto it, but it must be renamed for the next *Goma* run:

```
cp ec.int ec.exoII
```

Next, the input file for the next *Goma* run is prepared. Start with the given template:

```
cp input.template input
```

This file already contains all of the required information except for the starting time and time step. The last time step was 145611, so start the next run at 25% of this, or 36402. Find the string "dt" in the input file (`delta_t` card) and replace it with 36402. Specify the initial time by taking the final time from the previous run (436833) and replacing the string "tinit" (`Initial Time` card) with this value. This input file will then be ready for the next *Goma* run:

```
goma
```

Save the input and output files for this run segment with suffix t1:

```
cp ec.exoII ec.exoII.t1
cp ec.out.exoII ec.out.exoII.t1
cp input input.t1
```

--- END OF LOOP ---

To take additional steps, repeat each step between "START OF LOOP" and "END OF LOOP" above, incrementing the index (highlighted in red) each time.

To use a script to automatically perform this procedure, execute:

```
ec.serial.sh
```

or

```
ec.parallel.sh
```

Both scripts are set up to two remesh/remap/rerun steps after the initial run. Note that the scripts use GROPE to extract the last time and time step from the old output file, then call `sed` to insert the values into the new input file. The parallel script is set up to run Goma on 8 processors -- as noted above, the file can be edited to run on a different number of processors.

References

[1] A. El-Hamalawi, "A simple and effective element distortion factor." *Computers & Structures* **75**, 2000, 507-513.

[2] S. H. Lo, "Generating quadrilateral elements on plane and over curved surfaces." *Computers & Structures* **31**, 1989, 421-426.

[3] R. Sorenson, K. S. Chen, T. R. Guilinger, J. W. Braithwaite, & J. R. Michael. "Predicting the effects of corrosion on the performance of electrical contacts," in Electrochemical Society Proceedings Volume 2001-22, edited by J. D. Sinclair, R. P. Frankenthal, E. Kalman, & W. Plieth (2001).

[4] J. Braithwaite, N. R. Sorenson, D. Robinson, K. S. Chen, & C. Bogdan, "A modeling approach for predicting the effects of corrosion on electrical-circuit reliability," *Sandia Technical Report* SAND2003-0359, February 2003.

[5] K. S. Chen, "Goma modeling of atmospheric copper sulfidation with charge separation," *Sandia technical memorandum*, June 2003.

[6] K. S. Chen & G. H. Evans, "Two-dimensional modeling of nickel electrodeposition in LIGA microfabrication," paper presented at the *Fifth International Workshop on High Aspect Ratio Micro Structure Technology, Harmst '03*, Monterey, CA, June 15-17, 2003. Also submitted to *Microsystems Technology J.*, June 2003.