The General Purpose Parameter Based Two Dimensional Mesh Generator

Sivamayam Sivasuthan
Department of Electrical and Computer Engineering
Michigan State University
East Lansing, MI 48824, USA
Email: sivasuth@msu.edu

Abstract

In inverse electromagnetic problem solutions, parameter based mesh generation plays an essential role. As the geometry defined by parameters is optimized it changes shape, and a new finite element mesh must be created without stopping the optimization iterations to create a new mesh. Available mesh generators do not support such parameter based mesh generation without some manual input. This means that as optimization changes the shape we need to stop the program and make manual entry for a new mesh. The required mesh generator must therefore support parameter based mesh generation and be completely automatic once the optimization process begins – that means we must be able to change the physical shape of the problem during run time and generate the mesh. Such a mesh generator would help us implement a seamless optimization process in finite element analysis. This paper presents such an efficient, automatic, parameter-based mesh generator for optimization problems in finite element analysis. We present new software that can handle any general 2D geometrical shape and uses object oriented data structures to achieve superior performance. Examples of optimization carried out with this mesh generator are presented. Object oriented data structures are used to represent the problem. The software is developed in C/C++ and has been checked for many problems successfully.

Keywords: finite element analysis, Optimization, object oriented data structures, parameter based mesh generator

Introduction

One of the most popular methods in scientific computing is the finite element method. Mesh generation is a very important part of finite element analysis. There are many mesh generators available in the web and some of them are open source software [1 - 3]. These mesh generators do not support parameter based mesh generation but in real world problems (say inverse problems, i.e. optimization problems) we need parameter based mesh generators. In optimization the design is defined by parameters and an object function for minimization. As optimization proceeds, in each iteration the parameters are changed. For optimization to go on non-stop, the mesh needs to be generated for the new parameters. So far we have used problem specific mesh generators for these kinds of optimization problems. Problem specific meshes however give a

headache to the user; for every problem researchers and other users have to construct the problem specific meshes. The following paragraphs describe the necessity of parameter based mesh generators for inverse problems. Practically, inverse problems play an essential role in device design; say: compute the size and other descriptions of a motor that can produce so much torque

Figure 1 shows the design cycle for an inverse problem. In the first step the design parameter set h is randomly selected (or estimated by a subject expert) and thereupon we generate the parameter based mesh, get the finite element solution and measure the object value (often conveniently defined as a least square difference between design objects desired and computed) and check whether it is minimum or not. If this is minimum we terminate the loop; otherwise we change the design parameters and do the same procedure again. This procedure repeats until the object value becomes minimum [4].

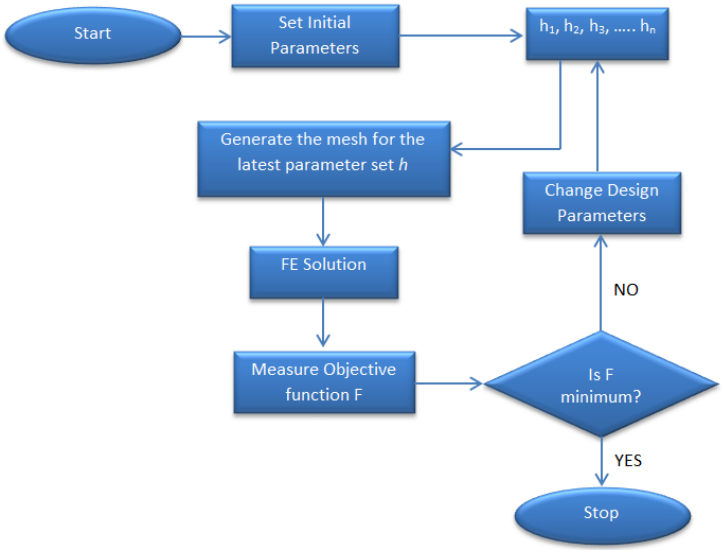

Figure 1: The Design Cycle for Inverse Problem

Background and Related Work

In the early 1970's, the Finite Element Method (FEM) was starting to play a major role in engineering design offices. Nowadays many researchers who are mainly in structural design and computational electromagnetics use different kinds of mesh generators for finite element analysis. For seamless iteration of inverse problems, we need parameter based mesh generators. A literature search shows no efficient open source software is available for parameter based mesh generation. Meshing is a discrete representation of the problem space. There are two types of two

dimensional cell shapes commonly used. There are the triangle and the quadrilateral. In this mesh generator we used the triangle cell. This cell consists of three sides.

Triangle[1] is a two dimensional mesh generator and Delaunay triangulator [5]. *Triangle* generates exact Delaunay triangulations, constrained Delaunay triangulations [1], conforming Delaunay triangulations, Voronoi diagrams, and high-quality triangular meshes. The latter can be generated with no large angles, and are thus suitable for finite element analysis [1]. In this work we used *Triangle* as a backend. Triangle is not a parametric mesh generator. It is for a single problem and requires manual input. Many modern mesh generators also used triangle as a backend but for a single solution at a time [7, 8].

New Mesh Generator

Input file format

Figure 2 shows the sample input file of our mesh generator adapting Triangle using a script file to run non-stop from iteration to iteration without manual intervention.

- Mesh generator code does not care about lines which start with #. We can write the comment using the # sign.

- First interpreted row: <a number, a number> - The first number represents the number of nodes in the domain; the second number represents the number of variable points. These variable points are also nodes but their coordinates may vary with the optimization iterations.

```
# A set of pointsin 2D(* WITHOUT VARIABLE POINTS).
# Number of nodes is 9 number of variables is 5
9 5
# And here are the nine points.
1 0.0 0.0
2 10.0 0.0
3 20.0 0.0
4 10.0 10.0
5 0.0 10.0
6 2.0 2.0
7 4.0 2.0
8 4.0 4.0
9 2.0 4.0
# variable points
# number of points in first draw. Then coordinates
10 20.0 3.0
```

Figure 2: Sample Input File for Mesh Generator

- From the second row to row number 10 (9+1) in the domain there are
  <a integer number, a floating point number, a floating point number > - The integer represents the node number. This must be numbered consecutively, starting from one. The two floating point numbers represent the coordinates of this node.

- The next segment of this input file represents variable points. This is also is in the same format as the previous.

- In the third part of the input file we have segment details. The first row of this file has the number of segments. From the second row it has 4 columns. The first row is the segment number which must be numbered consecutively, starting from one. The next two columns are node numbers. Each row represents a segment. The fourth column is a marker. A marker has different integer values; it can be used to define the boundary condition. Here -1 means it is not a boundary. If we have to define the boundary condition we have to give any positive integer to the marker. Then we can assign the boundary value using these markers.

- The next segment of this file is the definition of boundaries. The first row contains the number of boundary conditions. From the second row the first column represents the numbering; the second column is a marker number which has been already defined in the previous part (the segment part). The third column represents the boundary value for a particular marker.

- The subsequent segment is a definition of the regions of the problem. Here a different region means different materials so it has different properties. The first row represents the number of regions in the domain. From the second row, each row has five columns. The first column represents the numbering as usual. The second and the third columns represent the coordinates. These coordinates are used to identify the region. The point may be any point in the relevant particular region. The fourth column is an integer which starts from 1. It can be used to assign properties to these regions. The next column is not used here because it is an area constraint coming from Triangle but is not used by us.

- The next segment of this input file represents the number of holes. The hole is a region in which we do not want to generate the mesh. In the first step we define the number of holes in the problem domain. Form the next row, there are three columns. The first column represents the hole number. The second and third the columns represent the x and y coordinates of any point within that hole.

- The segment thereafter is for the measuring point list. The first part is the number of measuring points where we are going to calculate the solution to get the target solution

Our source code helps users to identify the errors in the input file. It works very efficiently for any shape of problem domain. The sample input file is attached in this paper as appendix. Users customize their own problem very efficiently as tried out in our lab [6]. This software is easy to

use. This software is well supported in any operating systems, i.e., Linux/Unix, Windows, MacOS etc.

Preprocess for FEM

The initial problem domain and its properties are obtained from the input file. These define,

- The number of points (number of points + number of variables)
- Point list (x coordinate and y coordinate)
- Number of segments
- Segment list
- Segment marker list (to determine the boundary)
- Boundary conditions
- Number of holes
- Hole list
- Number of region
- Properties of the regions
- Region list
- Measuring point list

After we triangulate this problem domain we have to define the boundaries and boundary values. Upon triangulation, we have triangle list (node numbers), property of regions and point list (for FE solution). We do not need to calculate the solution of known nodes so we have to separate the known and unknown nodes. The first step of preprocessing is renumbering the nodes. In this process, we

1) Define the boundary (generally using segment numbers)
2) Get the boundary values (different boundaries may have different boundary values)
3) Get all nodes which are on boundaries (We used the segment marker list to determine the boundary nodes)
4) Separate boundary elements and non-boundary elements; separate unknown nodes and known nodes
5) Give the first set of numbers for the unknown nodes and the last set of numbers for the known nodes
6) Renumber the whole point list based on new numbering.
7) Renumber the node entries in the triangle list based on the new numbering system.
8) Get all properties for the particular regions
9) Assign these properties to all corresponding triangles.

There are some post processes after we calculate the solution. There are,

1) Find the triangles that correspond to measuring points; that is get values in the object function. We have a set of measuring points. We have to find the corresponding

triangle for every measuring point and calculate the solution of each point. We already have obtained the solution of each node for each triangle.

2) Compare with the target solution and calculated solution and calculate the object function value using some optimization method.

Experiment and Result

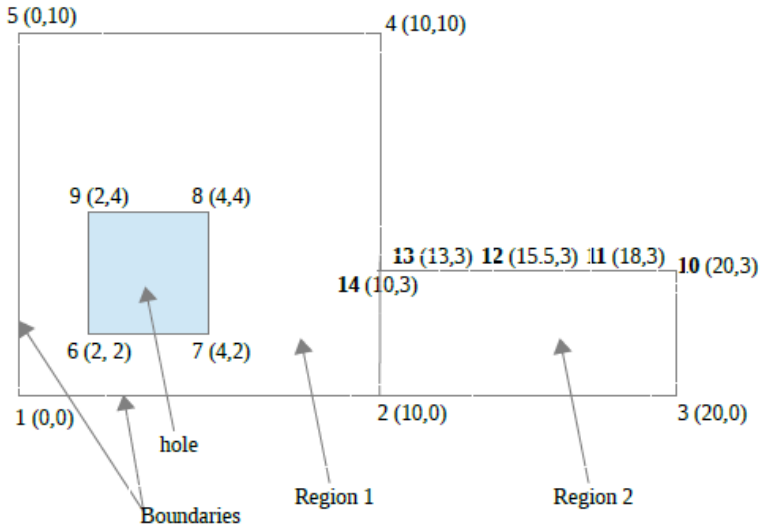

Figure 3: Sample problem

Figure 3 shows the example problem domain. There are 14 points (including 5 variable points 10 to 14), 2 regions, 1 hole, number of segments ((1, 2), (2, 3) etc) etc. The corresponding input file is attached in the last page. In this experiment, at every iteration we solved the Poisson equation ($-\varepsilon\nabla^2\varphi = \rho$) for this problem domain with arbitrarily imposed boundary conditions and changed the y value (known as design parameters) of each variable point with different values and generated the mesh seamlessly from one iteration to the next. Boundaries are given in Figure 3; the values of boundaries are 0.

(1)

Figure 4 shows the initial mesh for y coordinates for the variable nodes are 3.0. Figure 5 shows the corresponding equpotential lines. Figure 6 is the mesh for fifth iterative values; Figure 7 is the corresponding equpotential lines. Figure 8 is the mesh for fifth iterative values; Figure 9 is the corresponding equpotential lines.

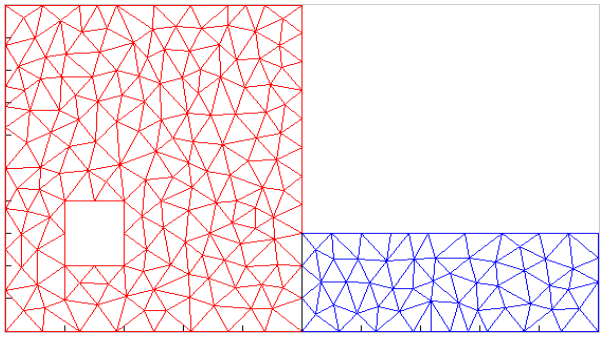

Figure 4: Initial Mesh

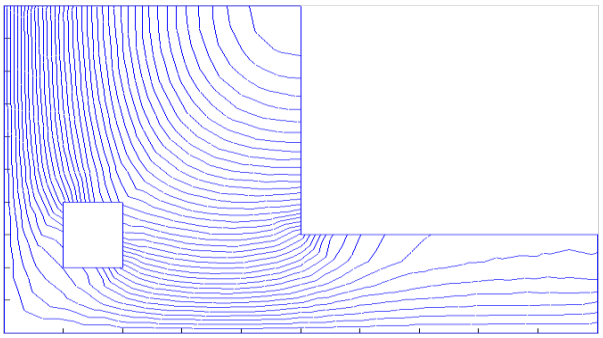

Figure 5: Equpotential line (initial)

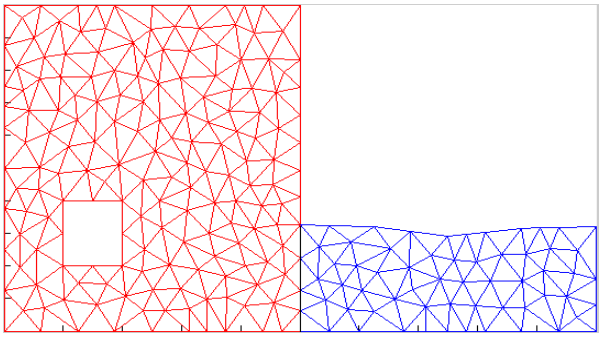

Figure 6: Mesh (After 5 iterations)

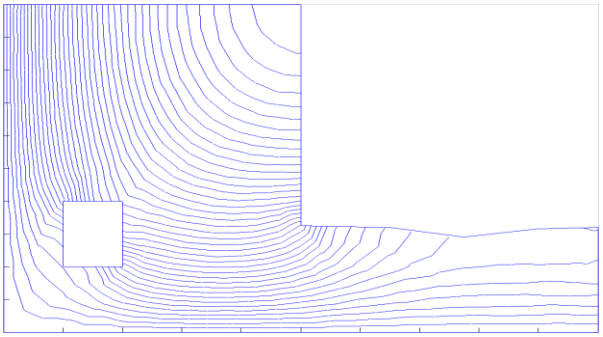

Figure 7: Equpotential line (after 5 its)

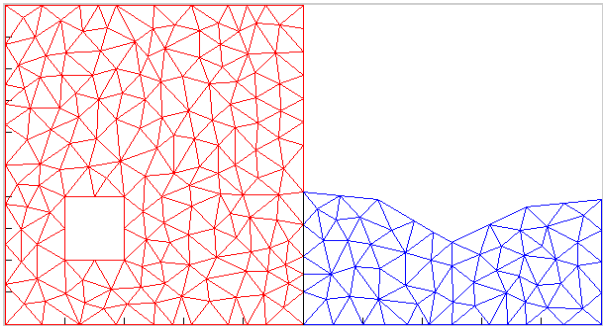

Figure 8: Mesh (After 10 iterations)

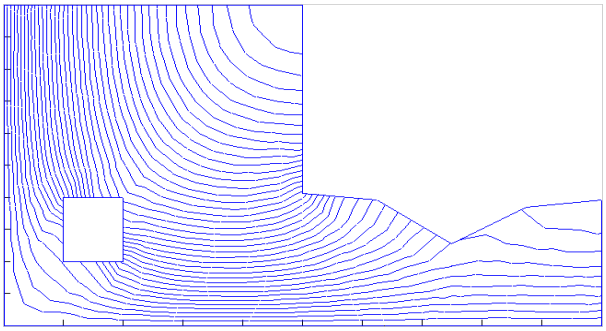

Figure 9: Equpotential line (after 10 its)

Conclusion

This paper describes the first successful general purpose parameter based two dimensional mesh generator software developed in C/C++ for seamless optimization iterations for Finite Element Design. The challenges of developing such software and the methods of overcoming those are discussed in detail, and its applications are explained using a sample problem.

Acknowledgements: Thanks to Professor S.R.H. Hoole of Michigan State University under whose guidance this project was carried out in his research lab and the high performance computing center (HPCC) facility at Michigan State University.

References

[1] Jonathan Richard Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, in "Applied Computational Geometry: Towards Geometric Engineering"' (Ming C. Lin and Dinesh Manocha, editors), volume 1148 of Lecture Notes in Computer Science, pages 203-222, Springer-Verlag, Berlin, May 1996.

[2] Jonathan Richard Shewchuk, Delaunay Refinement Algorithms for Triangular Mesh Generation, Computational Geometry: Theory and Applications 22(1-3):21-74, May 2002.

[3] Joachim Schoberl , NETGEN An advancing front 2D/3D-mesh generator based on abstract rules Computing and Visualization in Science Springer-Verlag 1997

[4] Sivasuthan, Sivamayam , Karthik Victor Uthayakumar , Hoole Samuel, "CUDA Memory Limitation in Finite Element Optimization to Reconstruct Cracks ". Review of progress in quantitative nondestructive evaluation. (2013).-in press

[5] Cendes, Z.J.; Shenton, D.; Shahnasser, H., "Magnetic field computation using Delaunay triangulation and complementary finite element methods," Magnetics, IEEE Transactions on , vol.19, no.6, pp.2551,2554, Nov 1983

[6] Karthik Victor Uthayakumar , Sivasuthan, Sivamayam , Hoole Samuel, "Parallel Implementation of the Genetic Algorithm on NVIDIA GPU Architecture for Synthesis and Inversion ". Review of progress in quantitative nondestructive evalutaion. (2013).- in press

[7] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering 79(11), pp. 1309-1331, 2009.

[8] Hang Si, TetGen: A Quality Tetrahedral Mesh Generator and 3D Delaunay Trianulator, Version 1.4 User Manual, Weierstrass Institute for Applied Analysis and Stochastics (WIAS), 16 January 2006

Appendix

```
# above shape
# Number of nodes is 9 number of variables is 5
9 5
# here nine points.
1 0 0
2 10 0
3 20 0
4 10 10
5 0 10
6 2 2
7 4 2
8 4 4
9 2 4
# variable points
10 20 3
11 18 3
12 15.5 3
13 13 3
14 10 3
```

```
# segments, 1st line --number of segments following lines are segments (node
numbers, each segment has two node numbers and a marker to identify the
boundary elements.
#number of segments
15
#segments (two nodes) and a marker
1     1     2     3
2     2     3     3
3     5     1     3
4     14    4     2
5     10    11    2
6     11    12    2
7     12    13    2
8     13    14    2
9     8     9     -1
10    9     6     -1
11    2     14    -1
12    3     10    -1
13    4     5     -1
14    6     7     -1
15    7     8     -1
#segment markers used to identify the boundaries and set boundary
#conditions. do not give 0 or 1 to segment marker because already
#fixed as a default, 3rd column is boundary condition value
2
1 2 0
2 3 100
#regions, number of regions x y coordinates of region, regional
#attribute (for whole mesh), Area constraint that will not be used
#we can leave one region without any assignments we have to assign for
#this case 0 0 0 0 but we can give properties to this region
2
1 1 0.5 1 0.1
2 12 3 2 0.9
#
#properties of regions, first number of properties then property
#values
2
1 1 1.32
2 1.90 9.312
# holes, number of holes x y coordinates of the hole
1
1 3 3
#number of measuring points
0
```