Circuit Simulation and Moving Mesh Generation

Gilbert Strang
Department of Mathematics, MIT
gs@math.mit.edu
http://math.mit.edu/˜gs

Per-Olof Persson
Department of Mathematics, MIT
persson@math.mit.edu
http://math.mit.edu/˜persson/mesh

***Abstract*— This paper outlines two different topics in computational engineering:**

1. **Many areas of applied mathematics lead to the same fundamental problem in numerical linear algebra. We will identify this problem for a network of resistors. Then we discuss the nonlinear problems of *circuit simulation*, and algorithms for solving them.**
2. **For partial differential equations (and also for computer graphics), an essential step is the generation of a well-spaced mesh. We describe a short, simple, and public code for unstructured meshes. Then we illustrate recent ideas in *moving the mesh* as the region changes.**

The third topic in Sapporo was wavelets. For a full exposition we refer to our textbook *Wavelets and Filter Banks* (Gilbert Strang and Truong Nguyen, Wellesley-Cambridge Press, 1996). A two-part Japanese translation of this book is published by Baifukan. This book concentrates on the connections to signal processing and image processing, including the wavelet transforms chosen for JPEG2000. For recent articles and other works on this active topic we refer to the *Wavelet Digest* on the web.

With the space and time that is available, we want to emphasize the "fundamental problem of applied mathematics" and the new ideas in mesh generation. Our work on linear algebra (the essential subject!) appears on the course page web.mit.edu/18.06. Video lectures are also available on MIT's OpenCourseWare website ocw.mit.edu under Mathematics. We hope these will be helpful to the reader.

I. Fundamental Problems of Networks

We begin with a circuit of m linear resistors connecting n nodes. The $m+n$ unknowns are the voltages $x_1,\ldots,x_n$ (the potentials at the nodes) and the currents $y_1,\ldots,y_m$ in the edges. There can be *current sources* $f_1,\ldots,f_n$ or *voltage sources* $b_1,\ldots,b_m$ (batteries in series with the resistors) or both.

In this very classical problem, we emphasize the "3-step framework", see Fig. 1. The material properties (eventually they become nonlinear) are in Ohm's Law $y = C(b - Ax)$. This involves each conductance c_i = (1/resistance) in the diagonal matrix C. Kirchhoff's Current Law is $A^{\mathrm{T}}y = f$. We can combine these equations using a block matrix of size $m+n$:

Fundamental Problem
$$\begin{bmatrix} C^{-1} & A \\ A^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix} \qquad (1)$$

The question is how to understand (and solve) this linear system. It appears everywhere in applied mathematics—we add more examples below. The block matrix has m positive pivots and eigenvalues (and n negative). We use m pivots in eliminating y to reach the symmetric matrix $A^{\mathrm{T}}CA$:

$$\begin{bmatrix} C^{-1} & A \\ A^{\mathrm{T}} & 0 \end{bmatrix} \to \begin{bmatrix} C^{-1} & A \\ 0 & -A^{\mathrm{T}}CA \end{bmatrix}. \qquad (2)$$

Eliminating y leaves

$$A^{\mathrm{T}}CAx \;=\; A^{\mathrm{T}}Cb - f. \qquad (3)$$

$K = A^{\mathrm{T}}CA$ has the great advantage that it is *positive definite*. In the finite element method, K is the "stiffness matrix"! Finite element codes almost always work with this displacement method, solving first for the unknowns x at the nodes. Then $y = C(b - Ax)$ gives the currents or edge forces.

We mention *Modified Nodal Analysis* [12], which simplifies the treatment of voltage sources (also op-amps and transformers and dependent sources). Variants of MNA are widely applied in simulation packages like SPICE.

There are three overall approaches to the fundamental equation (1):

A. Eliminate y and solve $Kx = A^{\mathrm{T}}Cb - f$ (with boundary conditions).

B. Solve $A^{\mathrm{T}}y = f$ first. The solution is not unique for $m > n$. We need a basis $v_1,\ldots,v_{m-n}$ for the solutions to $A^{\mathrm{T}}v = 0$. Then $y = y_0 +$ any combination $\sum z_i v_i$. In a network the vectors v_i come from *loops* in the graph. This could be effective if $m - n$ is small compared to n.

C. Solve the block system directly, usually by a direct elimination up to $n = 10^4$ or 10^5. A Krylov subspace iteration with preconditioner is faster with less storage for very large n. (The mixed method in finite elements, approximating both stresses y and displacements x, needs an "inf-sup" or "Babuska-Brezzi" condition to ensure that the block matrix is uniformly invertible.)

Figure 2 shows the same framework in other applications.

The last figure for nonlinear materials applies to transistors and diodes. The matrices for circuit simulation become unsymmetric and very large! Nevertheless they are extremely sparse. If ordered properly, their LU factors also remain remarkably sparse. We report here what we learned from very helpful discussion with Tim Davis.

The key is in *reordering the rows* to achieve two goals. First, it is useful to have a nearly zero-free diagonal (this is a maximal matching graph problem). Then the good feature of circuit matrices is that a permutation of rows and columns

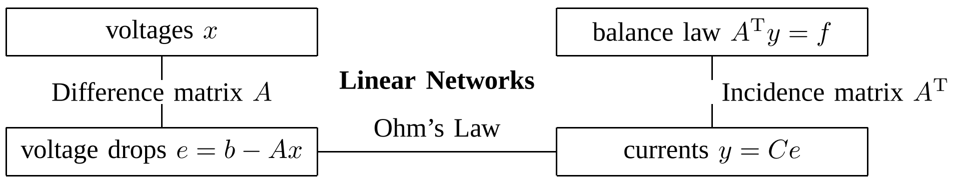
voltages x
balance law $A^{\mathrm{T}}y = f$
Difference matrix A
Linear Networks
Incidence matrix A^{T}
Ohm's Law
voltage drops $e = b - Ax$
currents $y = Ce$

Fig. 1. The "3-step framework" for circuit simulation.

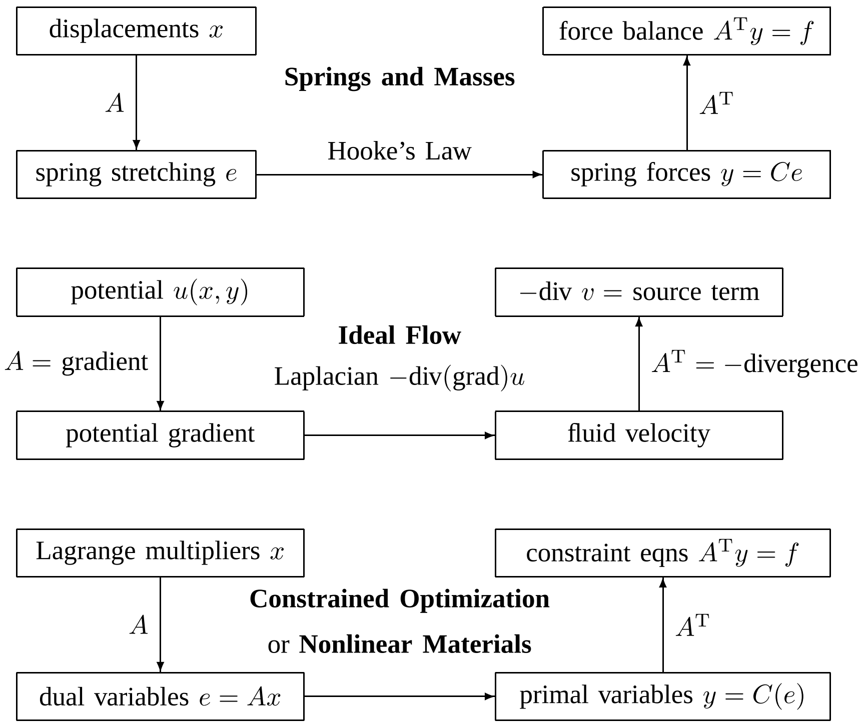

Fig. 2. The framework in other applications, leading to $A^TC(Ax) = f$.

produces a nearly *block triangular form*. This is important. Within each block, an approximate minimum degree (AMD) algorithm orders the rows to reduce fill-in for LU factorization.

In circuit simulation problems, fill-in often multiplies the original number of nonzero entries by *less than* 4. Then direct methods are very successful beyond $n = 10^6$. We are concentrating here on computing the DC operating point for analog circuits. (The full problem is a differential-algebraic equation, and the code moves from the DC point to compute transients.) Parasitic elements from capacitance between closely placed wires will harm the block triangular form and increase fill-in.

For modeling transistors with many parameters, parallel machines are valuable. The sparse "KLU" solution algorithm developed by Davis and Stanley (cise.ufl.edu/~davis) runs well on serial machines. For very large n, Sandia has developed the Xyce parallel circuit simulator, which adds preconditioning by incomplete (ILU) factorization. It includes hypergraph partitioning to scale up to distributed memory platforms. Please see the abstracts on www.us.sandia.gov/nacdm.

II. The Generation of Moving Meshes

In [5] we presented a new iterative mesh generation technique for implicit geometry representations. The inputs to our algorithm are the signed distance function $d(\boldsymbol{x})$ to the boundary (negative inside and positive outside), and a mesh size function $h(\boldsymbol{x})$ which gives the desired size of the elements. Assuming a piecewise linear force-displacement relationship in the mesh edges, we find an equilibrium position for the nodes. The mesh points that leave the domain during an update are projected back using the distance function. We showed how to implement the algorithm in a few dozen lines of MATLAB code, and the procedure tends to produce high quality meshes.

Here, we describe how to use our algorithm to mesh geometries that change with time ("moving meshes"). The iterative formulation is particularly useful for moving meshes since a good initial configuration is given at each step by the mesh from the previous step. Typically we only need a few additional iterations per step to obtain very good meshes. Also, since our mesh generator is based on distance functions we can use the *level set method* to propagate the geometry

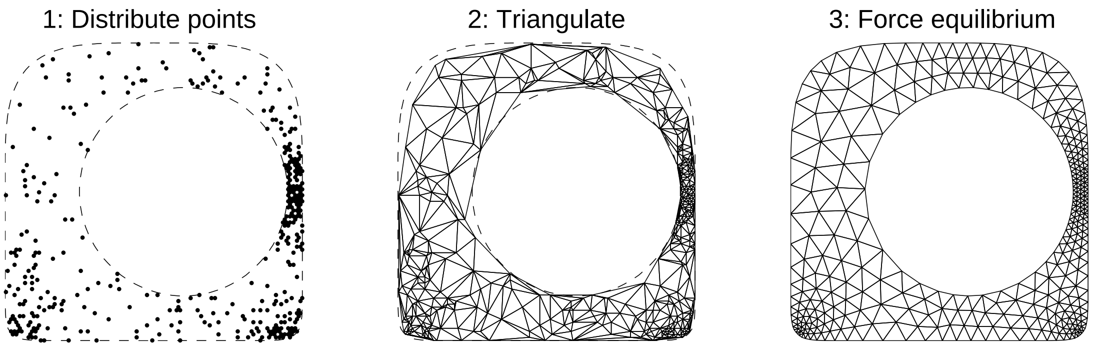

Fig. 3. The generation of a non-uniform triangular mesh, by force equilibrium at each node.

according to a given velocity field. In this way we get the benefits of the level set method (robust interface propagation, entropy solutions, topology changes, easy extension to higher dimensions) combined with the flexibility of general purpose finite element calculations on unstructured meshes.

Our moving algorithm uses a signed distance function $\phi(\boldsymbol{x})$ discretized on a Cartesian grid to represent the geometry, and bilinear interpolation to evaluate $\phi(\boldsymbol{x})$ for general $\boldsymbol{x}$. The geometry boundary is given by $\phi(\boldsymbol{x}) = 0$, and $\phi(\boldsymbol{x}) \leq 0$ inside the geometry. The initial distance function can be computed by explicit distance calculations close to the geometry boundary and the fast marching method for the rest of the domain. For simple initial geometries, $\phi(\boldsymbol{x})$ might also be available in a closed form expression.

The geometry boundary is then evolved in time according to a velocity field $\boldsymbol{v}(\boldsymbol{x})$ or a normal velocity field $T(\boldsymbol{x})$. These typically depend on the current (and the previous) geometries. In two-phase flow, the velocities are obtained by solving the Navier-Stokes equation using the distribution of the two fluids that is given by $\phi(\boldsymbol{x})$. The actual propagation of the boundary is done using the level set method, which solves hyperbolic PDEs on the Cartesian grid using entropy satisfying numerical schemes. For a velocity field $\boldsymbol{v} = (v_x, v_y)$ it solves the convection equation

$$\phi_t + \nabla\phi \cdot \boldsymbol{v} = 0 \tag{4}$$

and for a normal field T it solves the *level set equation*

$$\phi_t + T|\nabla\phi| = 0 \tag{5}$$

(both $\boldsymbol{v}$ and T may depend on space and time). These equations are solved using numerical discretizations, see [6] and [2] for details. After evolving ϕ, it does not in general remain a signed distance function. We *reinitialize* in the same way as we compute the initial distance function, by explicit updates of the distances close to the boundary $\phi(\boldsymbol{x}) = 0$ and the fast marching method.

We now turn to the generation of unstructured meshes for the sequence of discretized distance functions. At each step, our meshing algorithm needs an initial guess for the location of the mesh points. In [5] we used a random technique based on the rejection method to obtain a point density according to the size function $h(\boldsymbol{x})$ (Fig. 3, left). The mesh elements (the connectivity) is then found using the Delaunay algorithm (Fig. 3, center). We have also implemented a routine based on local refinements of an initial coarse mesh, which gives more stable results than the randomized method for complex geometries. For our moving meshes, this only needs to be done for the initial geometry. For all the other meshes, a fast start is obtained by displacing the mesh points a distance $\boldsymbol{v}(\boldsymbol{p})\Delta t$ for each mesh point $\boldsymbol{p}$.

To improve this initial mesh, we assign forces in the mesh edges and solve for force equilibrium at the nodes. The force in an edge depends on the length ℓ of the edge and on its unstretched length ℓ_0 (which we set proportional to the desired mesh size $h(\boldsymbol{x})$ evaluated at the edge midpoint). We use a linear spring model to push nodes *outward*:

$$f(\ell, \ell_0) = \begin{cases} k(\ell_0 - \ell) & \text{if } \ell < \ell_0, \\ 0 & \text{if } \ell \geq \ell_0. \end{cases} \tag{6}$$

By summing the forces at all mesh positions $\boldsymbol{p}$ (for each coordinate direction) we obtain a nonlinear system of equations $\boldsymbol{F}(\boldsymbol{p}) = 0$. We find the positions as a steady-state of

$$\frac{d\boldsymbol{p}}{dt} = \boldsymbol{F}(\boldsymbol{p}), \qquad t \geq 0 \tag{7}$$

using forward Euler. Note that this artificial time-dependence is unrelated to the (real) time evolution of the geometry as given by $\phi(\boldsymbol{x})$. After each Euler step we apply normal boundary forces, by projecting boundary points back orthogonally to the boundary using the distance function:

$$\boldsymbol{p} \leftarrow \boldsymbol{p} - \phi(\boldsymbol{p})\nabla\phi(\boldsymbol{p}). \tag{8}$$

These normal forces may be seen as Lagrange multipliers which keep the nodes exactly along the boundary. This expression can be modified to allow general implicit functions instead of distance functions, either by solving a system of nonlinear equations for each point (see [5]) or by approximate first- and second order projections.

During the iterations, we always maintain a good connectivity by updating the triangulation. In the simple MATLAB code of [5] this was done by recomputing the Delaunay triangulation, but we have implemented more efficient and

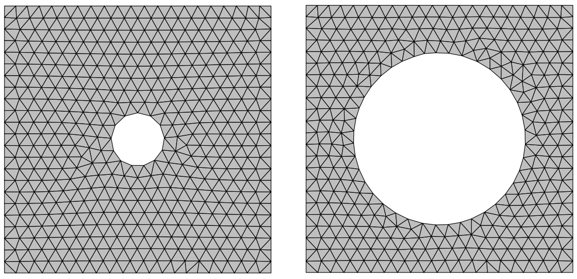

Fig. 4. Example of moving mesh, shown at two different times. The point density control is used to keep the mesh size uniform when the area changes.

robust versions based on local topology updates (such as edge flips). When the mesh quality is sufficiently high we terminate, and the mesh elements tend to form a mesh of high quality (Fig. 3, right).

The mesh size function $h(\boldsymbol{x})$ is important for generation of good meshes. It should specify smaller elements at boundaries of high curvature, regions of small feature size, or any other size constraints given by a numerical adaptive solver. In addition, the mesh size should not differ too much between neighboring elements, which corresponds to a limit on the gradient $|\nabla h(\boldsymbol{x})|$. We have developed techniques for automatic generation of mesh size functions from discretized distance functions. We compute boundary curvatures directly from $\phi(\boldsymbol{x})$, feature sizes by detecting the medial axis, and we limit $|\nabla h|$ by solving the *gradient limiting equation* [4]:

$$\frac{\partial h}{\partial t} + |\nabla h| = \min(|\nabla h|, g), \quad h(t=0) = h_0(\boldsymbol{x}). \tag{9}$$

In the examples below, we use uniform size functions ($h(\boldsymbol{x}) = \mathrm{constant}$) for simplicity. During the evolution of the geometry boundary, it is sometimes necessary to add or remove mesh points since $d(\boldsymbol{x})$ and/or $h(\boldsymbol{x})$ change with time. We do this by performing a density control at each time step, where we split an edge if it is too long compared to the desired value, and merge neighboring nodes if the edge is too short.

Moving meshes are best visualized as animations. Please visit math.mit.edu/~persson/mesh to view our movies. For the illustrations here, we show meshes at a few different times. As a simple example of a moving mesh, we study a geometry consisting of a square having a circular hole with a radius that changes with time, see Fig. 4. This geometry is easily written explicitly for all times, and we do not need to evolve the curve using the level set method. Note how the density control ensures that the element sizes are approximately equal even though the geometry area changes drastically.

One benefit of the algorithm is that mesh elements far away from the moving interface are left essentially unmodified. This gives several benefits, such as easier and more accurate solution transfer between the meshes and better opportunities for mesh compression. We also take advantage of this fact to improve the performance of our algorithm. We assign a stiffness to each mesh edge (the constant k in (6)) that increases with the distance from the moving interface. A few mesh elements away we set k to infinity, which means

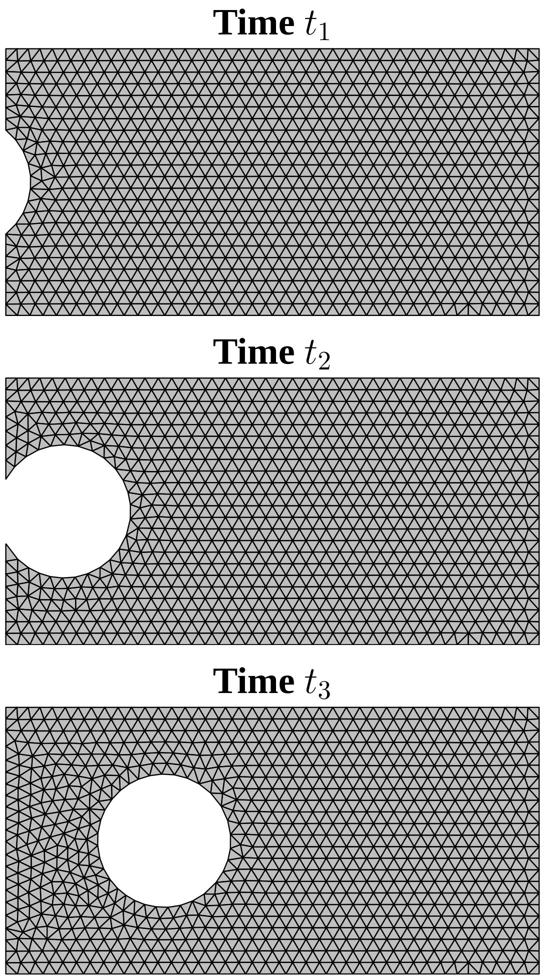

Fig. 5. Only mesh points close to the moving interface are allowed to move. This improves the performance dramatically and provides easier solution transfer between the old and new grids.

these nodes do not move at all. We can then ignore them when solving for force equilibrium, and this gives a dramatic performance improvement. The technique is illustrated by the example in Fig. 5, where we mesh a circular hole moving through a rectangle. Only a thin layer of elements close to the circle are allowed to move at each step, but the element qualities remain very high.

We now show examples that use discretized distance functions and the level set method to represent the moving geometries. There are many application areas and here we will focus on two shape optimization calculations. The first example comes from structural vibration control, and it was solved by Osher and Santosa using level set techniques on Cartesian grids [3]. We consider the eigenvalue problem

$$-\Delta u = \lambda \rho(\boldsymbol{x}) u, \qquad x \in \Omega \tag{10}$$

$$u = 0, \qquad x \in \partial\Omega \tag{11}$$

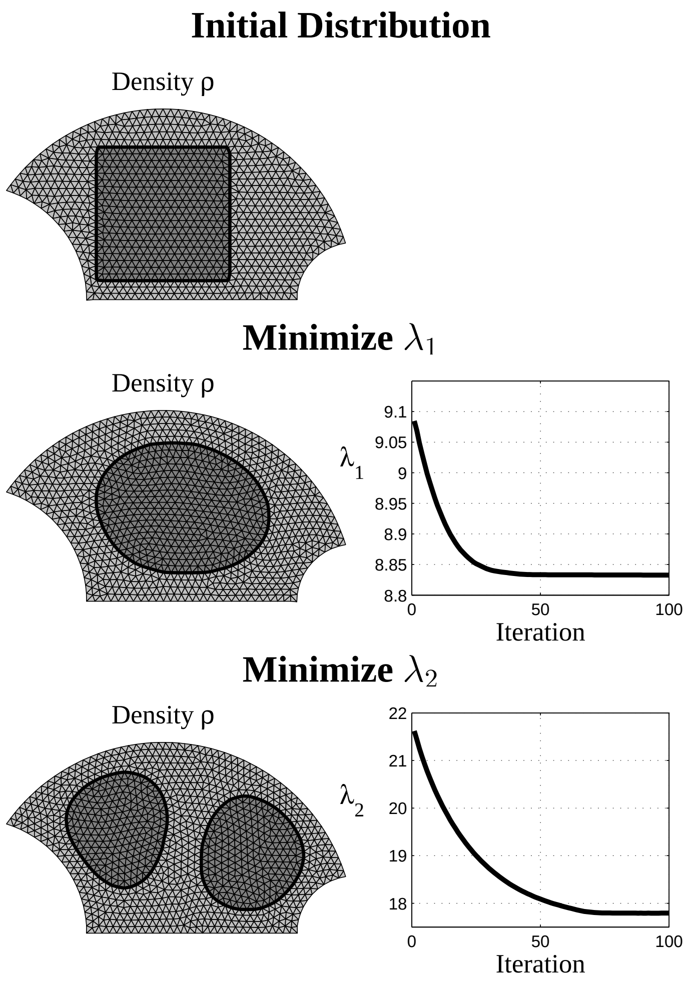

Fig. 6. Finding an optimal distribution of two different densities (light gray indicates low density) to minimize eigenvalues.

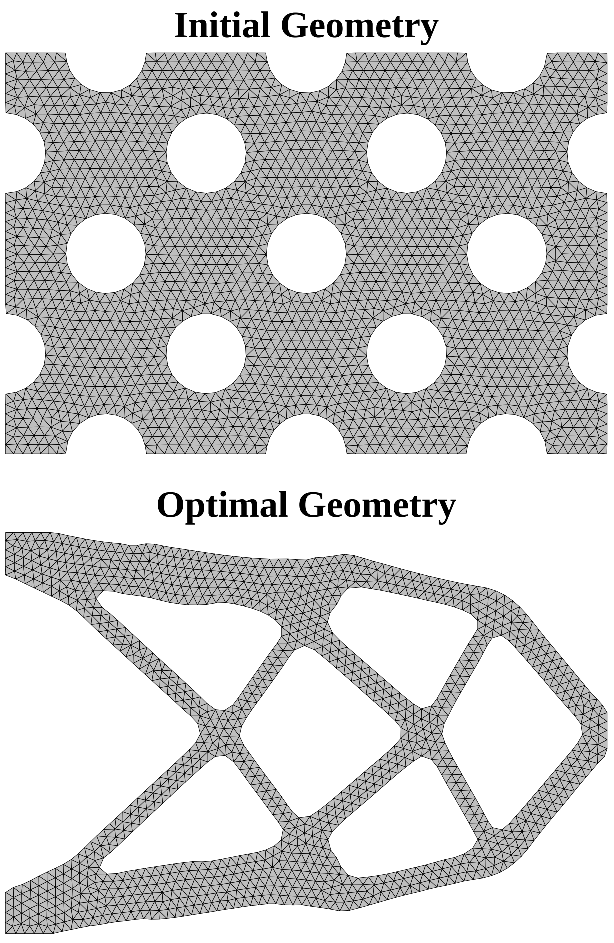

Fig. 7. Structural optimization for minimum of compliance. The structure is clamped at the left edge and loaded vertically at the right edge midpoint. Note how the initial topology is modified by the algorithm.

with

$$\rho(\boldsymbol{x}) = \begin{cases} \rho_1 & \text{for } x \notin S \\ \rho_2 & \text{for } x \in S, \end{cases} \tag{12}$$

and we minimize λ_1 or λ_2 subject to $\|S\| = K$.

We represent the boundary of S by a signed distance function on a Cartesian grid. To find the optimal distribution $\rho(\boldsymbol{x})$, we mesh the region (both inside and outside), solve the eigenvalue problem using finite elements on our unstructured mesh, and propagate the interface in the descent direction $\delta\phi = -v(\boldsymbol{x})|\nabla\phi|$ calculated using the current solution λ_i, u_i. To satisfy the area constraint we find a Lagrange multiplier using Newton's method.

Figure 6 shows the minimization of the first and the second eigenvalue. Note how the dark region is split into two separate regions in minimizing λ_2. This automatic treatment of topology changes is one of the main benefits of the level set method. By using unstructured meshes and the finite element method we achieve the following additional benefits:

- The material discontinuity is handled with high accuracy since the mesh fits to the interface between the two densities.
- We handle arbitrary outer geometries, again with high accuracy. Normally the level set method is used only on rectangular grids.
- We could have used graded meshes for more efficient simulations.

Our last example comes from structural design improvement. The geometry in Fig. 7 is clamped at the left edge, and a vertical force is applied at the midpoint of the right edge. We solve a linear elastostatic problem and minimize the compliance

$$\int_{\partial\Omega} g \cdot u \, ds = \int_{\Omega} A\epsilon(u) \cdot \epsilon(u) \, dx \tag{13}$$

subject to the area constraint

$$\|\Omega\| = K. \tag{14}$$

Sethian and Wiegmann solved this problem using level set techniques together with the immersed interface method [7]. Allaire, Jouve, and Toader used a similar technique but solved the linear elastostatic problem using an Ersatz material approach [1]. Since we have our high-quality unstructured meshes at each iteration, we can solve the physical problem using the finite element method. The optimal structure is shown in the bottom plot of Fig. 7.

Again we see advantages with our general meshes. The Neumann conditions at most of the boundaries are handled easily and accurately with the finite element method. We could again have used graded meshes to resolve fine details better. Finally, the finite element method is better developed than finite difference methods for advanced elasticity calculations, providing specialized elements.

We are currently studying several application areas for our technique. Other shape optimization problems appear in photonics, where again we believe the unstructured meshes will be of great importance. We study physical simulation with free boundaries, such as multi-phase fluid flow and linear elastic rearrangement instabilities. We are also extending the technique to three dimensional problems, something we believe will be easy since both the level set method and our mesh generation method generalize naturally to any dimension.

REFERENCES

[1] G. Allaire, F. Jouve, and A.-M. Toader. A level-set method for shape optimization. *C.R. Acad. Sci. Paris, Ser. I*, 334:1125–1130, 2002.

[2] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2002.

[3] S. Osher and F. Santosa. Level set methods for optimization problems involving geometry and constraints I. Frequencies of a two-density inhomogeneous drum. *J. Comput. Phys.*, 171:272–288, 2001.

[4] P.-O. Persson. PDE-based gradient limiting for mesh size functions. In *Proc. of 13th Int. Meshing Roundtable*. Sandia Nat. Lab., September 2004.

[5] P.-O. Persson and G. Strang. A simple mesh generator in matlab. *SIAM Review*, 46(2):329–345, June 2004.

[6] J. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry*. Cambridge University Press, 1999.

[7] J. Sethian and A. Wiegmann. Structural boundary design via level set and immersed interface methods. *J. Comput. Phys.*, 163:489–528, 2000.

[8] G. Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[9] G. Strang. *Linear Algebra and Its Applications*. Academic Press (now Brooks/Cole), 1988. Japanese translation.

[10] G. Strang and G. Fix. *An Analysis of the Finite Element Method*. Wellesley-Cambridge Press, 1973; Baifukan, 1977.

[11] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996; Baifukan, 2002.

[12] L. M. Wedepohl and L. Jackson. Modifi ed nodal analysis: an essential addition to electrical circuit theory and analysis. *Eng. Science and Education Journal*, 11(3):84–92, 2002.