SANDIA REPORT
SAND2020-9978
Printed September 18, 2020

Sandia National Laboratories

Shape Optimization for Control and Isolation of Structural Vibrations in Aerospace and Defense Applications

Hardesty, Sean, Kouri, Drew P., Lindsay, Payton, Ridzal, Denis, Stevens, Brian L., and Viertel, Ryan

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

ABSTRACT

Among the main challenges in shape optimization is the coupling of Finite Element Method (FEM) codes in a way that facilitates efficient computation of shape derivatives. This is particularly difficult with multi-physics problems involving legacy codes, where the costs of implementing and maintaining shape derivative capabilities are prohibitive. There are two mathematically equivalent approaches to computing the shape derivative: the volume method, and the boundary method. Each has a major drawback: the boundary method is less accurate, while the volume method is more invasive to the FEM code. Prior implementations of shape derivatives at Sandia have been based on the volume method. We introduce the *strip method*, which computes shape derivatives on a strip adjacent to the boundary. The strip method makes code coupling simple. Like the boundary method, it queries the state and adjoint solutions at quadrature nodes, but requires no knowledge of the FEM code implementations. At the same time, it exhibits the higher accuracy of the volume method. The development of the strip method also offers us the opportunity to share some lessons learned about implementing the volume method and boundary method, to show shape optimization results on problems of interest, and to begin addressing the other main challenges at hand: constraints on optimized shapes, and their interplay with optimization algorithms.

Acknowledgment

This work was supported by the Laboratory Directed Research and Development (LDRD) Engineering Sciences Research Foundation, and the U.S. Air Force Office of Scientific Research, Optimization Program under Award NO: F4FGA09135G001. Thanks to David Day (1542) and Scott Mitchell (1442) for helping us sort out Voronoi area calculations, Gregory Bunting (1542) for lots of help and advice on software design and interfacing with the existing inverse code and the p-element capability, and Tim Walsh (1543) and Brett Clark (1543) for funding related development work that greatly increased the impact of this project.

This page intentionally left blank.

CONTENTS

LIST OF FIGURES

1. Introduction

The importance of shape in engineering is rather obvious: aside from material properties, there's not much else that can be changed. Using optimization, we can improve some quantitative measure of design performance until we achieve optimality in a mathematical sense: when subject to design constraints, no further improvement is possible. When optimization is applied to "shape," there is a range of different ways to describe the changes under consideration, and corresponding terminology. Sigmund [2] distinguishes between sizing, shape, and topology optimization; see Figure 1.

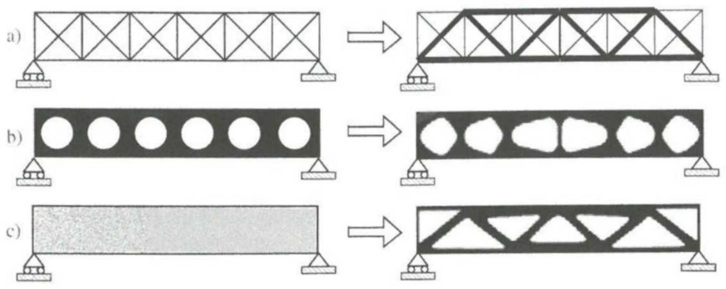

Figure 1. Figure from [2]. a) Sizing optimization can change a small set of parameters, e.g., thickness of beams, plates, etc. b) Shape optimization can make general changes to the shape. For example, a square could become a circle, but not a donut. c) Topology optimization can make general changes to both the shape and topology. The square can now become a donut.

Naturally, this distinction is somewhat artificial, as a design could be parametrized via some hybrid of the above approaches. This is in fact one of our long-term goals, but in order to understand the present state of the art, we must discuss shape and topology optimization as somewhat separate things. One might expect that efforts in this area would begin with the simplest approach and move towards the most general, but in fact, the opposite is the case: Sandia has invested substantially in topology optimization and has done essentially nothing for shape optimization (or even sizing optimization). Whether or not this is a sensible course of action, it is not an aberration. The often-cited shape optimization expert Martin Berggren describes the situation for shape optimization as follows [3]:

> *Although the field has developed and matured over the years, it is perhaps fair to say that the impact on science and engineering practice has been limited. In contrast, the technique of optimal layout of a linearly elastic structure using the material distribution method for topology optimization has, indeed, had a noticeable impact on the design of mechanical components.*

The simplicity of the material distribution method and its combination with well-behaved objectives such as compliance minimization that allow the use of specialized optimization algorithms have made it possible for topology optimization methods to solve a particular class of problems without the need to sort out some more basic issues required of a less specialized approach. Berggren continues:

One reason for the limited impact can be the complexity of managing a system for shape optimization: software for parametrization of shapes, mesh deformation, solvers, sensitivity analysis, and optimization needs to be developed and interfaced in an intricate way.

Although we must address all of these challenges, the most fundamental is that of shape sensitivity, which is the differential change in the solution with respect to the shape.

1.1. Shape Sensitivity

In order to use the gradient-based algorithms required to optimize over large numbers of shape parameters, shape sensitivity is a necessity. It has two mathematically equivalent formulations, which are related via integration by parts: the volume method (or "weak form"), and the boundary method (or "strong form"). The names can be confusing: the boundary method has nothing to do with boundary integral equations or Boundary Element Methods (BEM). The boundary method has a long history in the mathematics community, but has had minimal impact on engineering practice. This might be because foundational texts such as [19] are intimidating, or because practitioners simply were not aware of it. It was not until 2010 that Berggren noted the connection between the two methods [3], and not until 2015 that Hiptmair et al. [10] did the careful analysis required to understand the differing behavior of their discretizations. From the point of view of Numerical Analysis alone, the volume method is arguably superior because its discretization often exhibits a higher rate of convergence. However, this is not the most important difference between the two methods – a fact that still has not been recognized in the literature [17]. The volume method, and implementations thereof such as Automatic Differentiation (AD) [14] and Finite Differences (FD) are invasive: implementation of these methods requires modifications to the underlying FEM code because they need to differentiate FEM matrix entries with respect to shape. To do this for each type of finite element in Sandia's massive code base is prohibitively expensive. Jeff Cipolla, who is now at Thornton Tomasetti, but previously spent ten years working at ABAQUS, told us that they had an implementation of the volume method, but gave up due to the following difficulties:

- They became too dependent on a single developer, who was the only one with the specialized knowledge required to implement the volume method.
- The developer was unable to keep up with ongoing changes and additions to their FEM library.
- They concluded that it was better to use a "black box" approach, and make the optimization routines completely independent of the underlying FEM codes.

The boundary method instead requires data provided by the FEM code plus a collection of boundary geometry terms. These geometric terms come from the mesh, and are shared over different types of coupling and physics.

This project began as an effort to evaluate the behavior of the boundary method on problems of interest to Sandia, beginning with elastic vibration in vacuo. The idea was that reduced accuracy would be an acceptable trade-off for a method that is less invasive to the code, allowing us to do all kinds of previously impossible or intractable shape and topology optimization problems by leveraging our existing physics code base. Success would result in a capability unique to Sandia: commercial FEM codes have avoided the issue entirely by using a different method with limited applicability.

1.2. Optimality Criteria Methods

Many commercial FEM codes, such as ABAQUS, ANSYS, NASTRAN, and COMSOL currently provide shape and topology optimization capabilities. In addition to the intrinsic limitations of these capabilities, one major impediment to the applicability of these codes for shape or topology optimization at Sandia is their poor scaling performance to the enormous problem sizes required for design and qualification. Furthermore, their workflows often treat shape optimization as a post-processing step to topology optimization [2, 3]. This is a substantial limitation, as only small shape changes are allowed or expected. A comprehensive review would be inappropriate here, but as far as we can tell from publicly available information, these codes rely on FD or optimality criteria methods [13]. Although slow and inaccurate, FD is suitable for small parameter sets. For general "nonparametric" shape changes, optimality criteria methods use specific knowledge of the physical problem (e.g., to minimize tangential surface stresses) to avoid the need to compute sensitivities. In one of the foundational works, the trade-off for optimality criteria methods is described [7]:

> *The basic concept behind optimality criteria is the rejection of generality of mathematical programming and the utilization of the physical characteristics of the structural optimization problem to generate an approach of somewhat limited applicability, but of the greatest computational efficiency.*

In reality, the phrase "of somewhat limited applicability" is a serious understatement; e.g., optimality criteria methods, based on a simple fixed-point iteration, cannot deal with challenging, nonlinear shape constraints. At the same time, the phrase "of greatest computational efficiency" is a serious overstatement; there is no proof that the performance of the fixed-point iteration is independent of problem size, and there is no evidence that state-of-the-art optimization methods are inefficient. Therefore, we reject optimality criteria methods in favor of the shape sensitivity and modern optimization algorithms. We believe that shape optimization has a crucial role to play in design, and that it is not merely a post-processing step for topology optimization: by moving surfaces without considering topological changes, shape optimization reduces the dimension of the design space, mitigating problems posed to optimization algorithms by poor conditioning and local minimizers. Our approach will have excellent performance and general applicability. The research question addressed in this work is to find a shape sensitivity method that is sufficiently accurate and is not invasive to FEM codes.

1.3. Shape Optimization Logistics

The ability to compute shape gradients is essential to a workable scheme for shape optimization, but it is not sufficient. The quotation from Martin Berggren in Section 1 mentions two other crucial ingredients: a means of updating the mesh as the domain is deformed, and the issue of how to constrain and parametrize the design.

General shape optimization problems require constraints on surface smoothness, and may also use constraints to enforce symmetry or manufacturability. Stress constraints or other state constraints that aren't purely geometric may also be in play. Simple user specification of useful types of constraints is a research question, although the methods used in commercial software can provide a guide [6]. An additional class of constraints that seems not to be considered within the framework of small shape changes is that of contact constraints, i.e., constraints to prevent disjoint pieces of the structure from coming into contact as the shape is changed. Naturally, contact constraints are unnecessary for topology optimization. These constraints will be non-linear, and need to be differentiable. In aggregate, all of these constraints require an advanced optimization algorithm, capable of handling large numbers of variables, scaling issues, and a variety of equality and inequality constraints.

Shape optimization that makes substantial changes to the location of boundary vertices must update the mesh in order to maintain sufficient element quality [3, 13]. We favor methods that retain a roughly constant element size as the boundary moves, rather than simply stretching existing elements with harmonic functions or smoothing. This requires the capability to add or delete elements, and good algorithms can be implemented with the existing STK software [21] with only moderate effort [11].

The issue of updating the mesh is tied in a surprising way to the choice of the algorithm to be used for shape sensitivity. Discretizations of the boundary method provide gradient values on the domain boundary only. Discretizations of the volume method, however, provide gradient values throughout the domain. Some immediate questions come to mind: what is to be done with the interior gradient values? Can they be used to update the mesh? Our work in shape sensitivity aims to provide satisfactory answers to these questions.

2. The Volume Method for Shape Sensitivity

To our surprise, the volume method turned out to be less difficult to implement than expected. The use of test-driven development techniques, and the already modular nature of the Sierra/SD [4] code base led us to the realization that most of the code required for elasticity and acoustics is in fact independent of the specific element type being used. Only a single specialized function per element type is required. We started out by implementing the volume method for tetrahedral elements, and got standard hexahedral elements and subparametric P-elements almost for free. Scripts can easily be written using, e.g., Matlab, to do the symbolic manipulations required and write the associated C++ code. This section summarizes these results in the simpler case of the Laplace operator in two dimensions, tying the mathematical expressions developed in the discretization to their implementation in code.

2.1. Discretization

We consider the classical PDE-constrained shape optimization problem [5, 15, 19]. Let $\Omega \subset \mathbb{R}^N$, with $1 \le N \le 3$, be a bounded open set with a sufficiently smooth boundary $\partial\Omega$. We wish to solve the problem

$$\min_{\Omega} \mathscr{J}(\Omega) := \int_{\Omega} j(u)\, dx, \tag{2.1a}$$

where $\mathscr{J}$ is the shape functional, the integrand $j : \mathbb{R} \to \mathbb{R}$ has a locally Lipschitz continuous derivative j_u, and u in (2.1a) solves the PDE

$$\begin{cases} -\Delta u &= f \quad \text{in} \quad \Omega \\ \quad u &= 0 \quad \text{on} \quad \partial\Omega, \end{cases} \tag{2.1b}$$

Let $\{\varphi_i : i = 1, \ldots, N_h\}$ be a nodal Lagrange basis for the finite element subspace $\mathbb{V}_h$, consisting of piecewise-linear functions defined on a triangular mesh. Then with

$$u_h(x) = \sum_{i=1}^{N_h} U_i \varphi_i(x), \quad p_h(x) = \sum_{i=1}^{N_h} P_i \varphi_i(x),$$

the discrete state and adjoint equations can be written

$$KU = F \tag{2.2}$$

$$K^{\top} P = G, \tag{2.3}$$

where

$$\begin{aligned} K_{ij} &= \int_{\Omega} \nabla\varphi_i \cdot \nabla\varphi_j \, dx, \\ F_i &= \int_{\Omega} f \varphi_i \, dx, \\ G_i &= \int_{\Omega} j_u(u_h) \varphi_i \, dx. \end{aligned} \tag{2.4}$$

Assume that there are S_h nodal coordinates, which in two dimensions is $S_h = 2N_h$, corresponding to shape variables s_k, $k \in \{1,\ldots,S_h\}$. We will denote, e.g., the discrete shape derivative of K for $k \in \{1,\ldots,S_h\}$ by

$$(K')_{ijk} = \frac{\partial K_{ij}}{\partial s_k}.$$

Using this notation, the discrete shape derivative of (2.2) is

$$K'U + KU' = F',$$

which can be rearranged into the discrete shape sensitivity equation

$$U' = K^{-1}\left(F' - K'U\right),$$

The discrete volume shape gradient is then

$$G^\top U' = \underbrace{G^\top K^{-1}}_{=P^\top}\left(F' - K'U\right), \tag{2.5}$$

where we recall from (2.3) that $P^\top = G^\top K^{-1}$. To complete the description, we need to work out formulas for the discrete operator derivatives K' and F' in (2.5). It is sufficient to consider their local assembly, i.e., on a single triangle. Recall that the local contributions to K and F are given in (2.4).

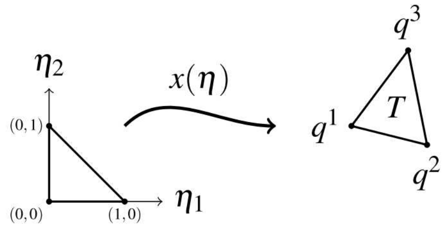

Figure 2. Mapping from unit triangle reference element with coordinates $\eta \in \mathbb{R}^2$ to a triangle T with vertices $q^1, q^2, q^3 \in \mathbb{R}^2$.

The mapping of the reference element to a mesh triangle with vertices $q^1, q^2, q^3 \in \mathbb{R}^2$ (see Figure 2) is given by

$$x(\eta) = q^1 + \underbrace{\left(q^2 - q^1 \quad q^3 - q^1\right)}_{\equiv J}\begin{pmatrix}\eta_1\\ \eta_2\end{pmatrix}. \tag{2.6}$$

We define the standard piecewise-linear basis functions via the mapping $\ell : \mathbb{R}^2 \to \mathbb{R}^3$ via

$$\ell(\eta) = \begin{pmatrix} 1 - \eta_1 - \eta_2 \\ \eta_1 \\ \eta_2 \end{pmatrix},$$

and express them in mesh coordinates via

$$\hat{\ell}(x(\eta)) = \ell(\eta).$$

We thus obtain the local quantities

$$\mathbb{R}^{3\times 3} \ni K_{\text{loc}} = \int_T \nabla\hat{\ell}\nabla\hat{\ell}^\top\,dx = \int_0^1\int_0^{1-\eta_1} \nabla\ell J^{-1}J^{-\top}\nabla\ell^\top \det J\,d\eta_2 d\eta_1, \tag{2.7}$$

$$\mathbb{R}^{3\times 1} \ni F_{\text{loc}} = \int_T \hat{\ell} f(x)\,dx = \int_0^1\int_0^{1-\eta_1} \ell f(x(\eta))\det J\,d\eta_2 d\eta_1, \tag{2.8}$$

recalling the definition of the Jacobian matrix J in (2.6). Differentiation of K_{loc} with respect to the six local degrees of freedom (two per vertex) on the triangle T is thus a matter of differentiating the terms J^{-1} and $\det J$. Differentiation of F_{loc} requires differentiation of the the argument $x(\eta)$ in $f(x(\eta))$, and differentiation of $\det J$.

Recall than J can be expressed (c.f. (2.6); the subscript denotes the first and second components)

$$J = \begin{pmatrix} q_1^2 - q_1^1 & q_1^3 - q_1^1 \\ q_2^2 - q_2^1 & q_2^3 - q_2^1 \end{pmatrix}, \tag{2.9}$$

so that

$$\det J = (q_1^2 - q_1^1)(q_2^3 - q_2^1) - (q_2^2 - q_2^1)(q_1^3 - q_1^1). \tag{2.10}$$

Next, we shall compute the shape variations of the $\det J$ and J^{-1}. Consider a perturbation $(\delta q^1, \delta q^2, \delta q^3)$ of the vertices (q^1, q^2, q^3) of an element T. The change in $\det J$ in (2.10) is given by

$$\begin{aligned}(\det J)'\delta q =& (\delta q_1^2 - \delta q_1^1)(q_2^3 - q_2^1) - (\delta q_2^2 - \delta q_2^1)(q_1^3 - q_1^1) + \\ & (q_1^2 - q_1^1)(\delta q_2^3 - \delta q_2^1) - (q_2^2 - q_2^1)(\delta q_1^3 - \delta q_1^1).\end{aligned}$$

Thus, if we arrange the local shape parameters into a vector

$$\mathbb{R}^6 \ni q = (\delta q_1^1, \delta q_2^1, \delta q_1^2, \delta q_2^2, \delta q_1^3, \delta q_2^3)^\top, \tag{2.11}$$

then we can express the discrete shape derivative via

$$\mathbb{R}^{1\times 1\times 6} \ni (\det J)' = \begin{pmatrix} (q_2^2 - q_2^1) - (q_2^3 - q_2^1) \\ (q_1^3 - q_1^1) - (q_1^2 - q_1^1) \\ (q_2^3 - q_2^1) \\ -(q_1^3 - q_1^1) \\ -(q_2^2 - q_2^1) \\ (q_1^2 - q_1^1) \end{pmatrix}. \tag{2.12}$$

For the matrix $J \in \mathbb{R}^{2\times 2}$, this gets a little harder to visualize. Using the same ordering (2.11), and recalling equation (2.9), we have

$$(J')_{ijk} = \frac{\partial J_{ij}}{\partial q_k}, \tag{2.13}$$

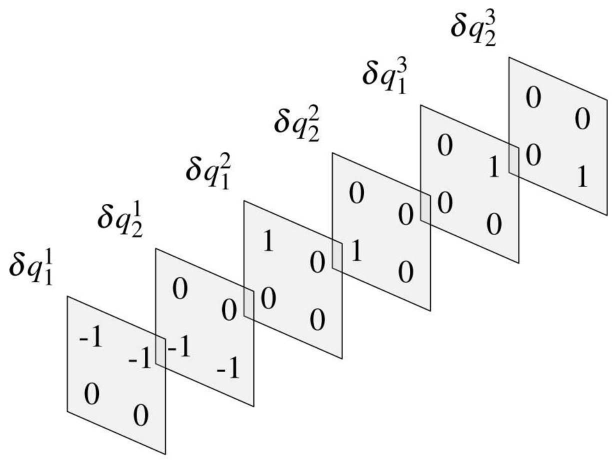

Figure 3. The discrete shape gradient $J' \in \mathbb{R}^{2\times2\times6}$.

which is depicted in Figure 3. Recalling that we required the shape derivative of J^{-1} rather than of J, we must make use of the identity $JJ^{-1} = I$:

$$J'J^{-1} + J(J^{-1})' = 0,$$

which implies that

$$(J^{-1})' = -J^{-1}J'J^{-1}. \tag{2.14}$$

These Jacobian transformations are applied to local state and adjoint vectors before contraction against J'. It then remains to translate the six components corresponding to the local shape parameters into a global gradient vector.

2.2. Implementation

In Section 2.1, we identified two main shape gradient operations related to the Jacobian matrix (2.6): $(\det J)'$, and $(J^{-1})'$. Both of these can be expressed in an element-independent way in terms of the shape gradient of the Jacobian itself. We can thus write the following functions:

- `JacobianDeterminantShapeGradient` to compute $(\det J)'$, the shape gradient of the Jacobian determinant (2.12).
- `JacobianInverseShapeGradient` to compute $(J^{-1})'$, the shape gradient of the inverse Jacobian matrix (2.14).

These functions then call a C++ virtual function `JacobianShapeGradient`, which must be implemented for each element type. It implements J', which is depicted in Figure 3 for a piecewise-linear triangle. For more general element types, this operation becomes more complex; the entries depend in general on the quadrature point, but this part can be completely separated from the implementation of `JacobianDeterminantShapeGradient` and `JacobianInverseShapeGradient`, both of which must call `JacobianShapeGradient`.

3. The Boundary Method for Shape Sensitivity

The boundary method is known to be less accurate than the volume method for computing shape derivatives [10]. We demonstrate via a numerical experiment in Section 3.1 that this reduced accuracy can in fact cause optimization algorithms to converge much more slowly. The optimistic interpretation of this result is to note that the optimization still converges to a plausible answer. However, determining appropriate criteria for convergence is challenging in general, and at any rate, there is a more serious problem.

The premise of this research was that the boundary method could be a viable alternative to the volume method despite reduced accuracy because it is not invasive to the code. Although the idea that it does not require direct modification of the code assembling FEM matrices is correct, we found that it does require information that was not readily available in Sierra/SD; in Section 3.2, we show that evaluation of the boundary method for general elasticity problems requires evaluation of the stress and strain on the domain boundary. Prior to this project, Sierra/SD provided the stress and strain only at element centroids, but not on the surface. To provide the capability to evaluate stress and strain on the surface was a long-standing request of analysts, who value the greater accuracy with which peak values can be determined when surface evaluation is available. Thus, the cost of implementing surface stress and strain evaluation was split equally between this project and other funding. The upshot was that we provided a useful capability that is now being used in production analysis cases. On the other hand, the high cost relative to the original budget provided for this project undercuts the notion that the boundary method can be implemented cheaply and in generality; the mathematical manipulations shown in Section 3.2 that have to be repeated in general are also a significant part of the cost of the boundary method.

3.1. Accuracy

We have discretized the optimization problem (2.1), choosing the domain shape, objective function, and boundary conditions in such a way as to make the problem challenging for the boundary method. Note that our model problem (2.1b) has Dirichlet boundary conditions, which is known from [10, § 4] to be the more challenging case.

We have chosen the initial domain to be $\Omega_0 = (0,1)^2$ and aim to recover the circle Ω_* that circumscribes Ω_0:

$$\Omega_* = \left\{ x \in \mathbb{R}^2 \ : \ \left| x - \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \end{pmatrix}^\top \right|_2 < \frac{\sqrt{2}}{2} \right\}.$$

Turning the square into a circle via optimization requires smoothing out its corners, which poses challenges for mesh quality.

The integrand j in the objective function (2.1a) is chosen to be

$$j(u) = \frac{1}{2}(u - u_*)^2,$$

where

$$u_*(x) = J_0\left(\lambda \left| x - \begin{pmatrix}\frac{1}{2} & \frac{1}{2}\end{pmatrix}^\top \right|_2\right)$$

solves (2.1b) on Ω_* with forcing function

$$f(x) = \lambda^2 u_*(x).$$

In the above, J_0 is the order-zero Bessel function of the first kind, and λ is chosen so that the first zero of J_0 occurs at the boundary of the circle, i.e., the smallest positive real number such that

$$J_0\left(\lambda \frac{\sqrt{2}}{2}\right) = 0.$$

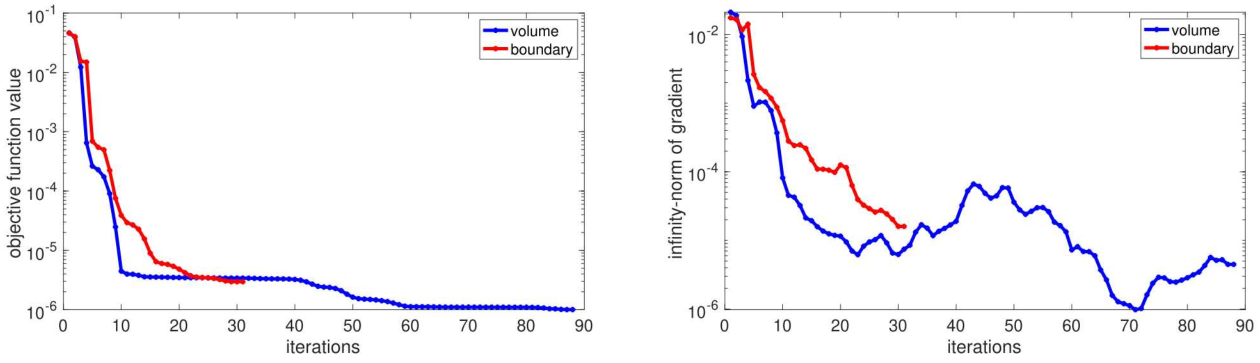

Figure 4. Convergence of the objective function and gradient values (∞-norm) for both the volume and boundary methods. The stopping criterion is a minimum step size of 10^{-10}. The volume method reaches the objective function value achieved by the boundary method much faster, and subsequently continues to reduce it, by a factor of three.

Results on this model optimization problem are shown in Figure 4. After about 10 iterations, the volume method reaches an objective function value comparable to the final objective value attained by the boundary method after 30 iterations. Moreover, the volume method continues to reduce the objective function until it is approximately three times smaller. Since the boundary method gradient is inconsistent with the discretization, we choose to stop at stagnation of the method rather than using a gradient tolerance.

3.2. Elasticity - Theory

This exposition uses the expression for the shape derivative of the solution to the linear elastostatics problem from [19, §3.5]. We combine this with a general definition of our PDE-constrained optimization problem and corresponding adjoint equation in order to derive the expression for the boundary shape gradient.

Let $\Omega \subset \mathbb{R}^3$, be a bounded open set with a sufficiently smooth boundary $\Gamma = \overline{\Gamma_0} \cup \overline{\Gamma_1}$, unit normal n, $\text{meas}(\Gamma_0) > 0$, $f \in L^2(\Omega; \mathbb{R}^3)$, and $p \in H^1(\Gamma_1; \mathbb{R}^3)$.

We consider the minimization problem

$$\min_{\Omega} \mathscr{J}(\Omega) := \int_{\Omega} j(u)\,dx, \tag{3.1a}$$

where $\mathscr{J}$ is the shape functional and the integrand $j : \mathbb{R}^3 \to \mathbb{R}$ has a locally Lipschitz continuous derivative j_u. Moreover, u in (3.1a) solves the state equation

$$\begin{cases} -\nabla\cdot\sigma(u) & = f \quad \text{in} \quad \Omega \\ u & = 0 \quad \text{on} \quad \Gamma_0 \\ \sigma(u)\cdot n & = p \quad \text{on} \quad \Gamma_1, \end{cases} \tag{3.1b}$$

where

$$\begin{aligned} \sigma(u) &= C : \varepsilon(u) \\ \varepsilon_{ij}(u) &= \frac{1}{2}(\partial_i u_j + \partial_j u_i) \\ (\nabla\cdot\sigma)_i &= \sum_{j=1}^{3} \partial_j \sigma_{ij}, \end{aligned}$$

and C satisfies appropriate conditions on symmetry, boundedness, and positive definiteness.

The weak form of (3.1b) is

$$\int_{\Omega} (C : \varepsilon(u)) : \varepsilon(\varphi) = \int_{\Omega} f\cdot\varphi + \int_{\Gamma_1} p\cdot\varphi, \quad \forall\varphi \in \{\varphi \in H^1(\Omega;\mathbb{R}^3) : \varphi = 0 \text{ on } \Gamma_0\}. \tag{3.2}$$

We define the adjoint equation

$$\begin{cases} -\nabla\cdot\sigma(q) & = j_u \quad \text{in} \quad \Omega \\ q & = 0 \quad \text{on} \quad \Gamma_0 \\ \sigma(q)\cdot n & = 0 \quad \text{on} \quad \Gamma_1, \end{cases} \tag{3.3}$$

with the corresponding weak form

$$\int_{\Omega} (C : \varepsilon(q)) : \varepsilon(\varphi) = \int_{\Omega} j_u\cdot\varphi + \int_{\Gamma} (\sigma(q)\cdot n)\cdot\varphi, \quad \forall\varphi \in H^1(\Omega;\mathbb{R}^3). \tag{3.4}$$

The set $\mathscr{D}^1 := \mathscr{D}^1(\mathbb{R}^3;\mathbb{R}^3)$ consists of functions that are continuously differentiable in $\mathbb{R}^3$ with compact support. For a shape variation $v \in \mathscr{D}^1$ such that $v = 0$ on $\overline{\Gamma_0} \cap \overline{\Gamma_1}$, the shape sensitivity equation [19, (3.160)-(3.162)] is

$$\begin{cases} -\nabla\cdot\sigma'(u) & = 0 \quad \text{in} \quad \Omega \\ u' & = -v_n \frac{\partial u}{\partial n} \quad \text{on} \quad \Gamma_0 \\ \sigma'(u)\cdot n & = v_n f + v_n \kappa p + \operatorname{div}_{\Gamma}(v_n \sigma_\tau) \quad \text{on} \quad \Gamma_1, \end{cases} \tag{3.5}$$

where $v_n = v \cdot n$. Note that there is a typo in [19, (3.162)]. The trouble begins in (3.157), which should read:

$$\begin{aligned}&\int_\Omega \sigma' : \varepsilon(\phi) dx = -\int_\Omega \mathrm{div}\sigma' \cdot \phi + \int_\Gamma n \cdot \sigma' \cdot \phi d\Gamma = \\ &\int_\Gamma \{\underbrace{\phi \cdot \mathrm{div}_\Gamma(v_n \sigma) - v_n \kappa n \cdot \sigma \cdot \phi}_{(3.156)} + \underbrace{v_n f \cdot \phi + v_n \kappa \cdot p \cdot \phi}_{(3.155)}\} d\Gamma\end{aligned}$$

The term coming from (3.156) in the second line of (3.157) has its sign flipped in the original text. Following it through correctly, the sign of the div_Γ term in (3.162) flips. It should read:

$$\sigma' \cdot n = v_n f + v_n \kappa p + \mathrm{div}_\Gamma(v_n \sigma_\tau)$$

The weak form of the sensitivity equation (3.5) is

$$\begin{aligned}\int_\Omega (C : \varepsilon(u')) : \varepsilon(\varphi) &= \int_{\Gamma_1} v_n f \cdot \varphi + v_n \kappa p \cdot \varphi + \mathrm{div}_\Gamma(v_n \sigma_\tau) \cdot \varphi && (3.6)\\ &= \int_{\Gamma_1} v_n f \cdot \varphi + v_n \kappa p \cdot \varphi - v_n \sigma_t : \nabla_\Gamma \varphi. && (3.7)\end{aligned}$$

The gradient calculation proceeds as follows:

$$\begin{aligned}\int_\Omega j_u \cdot u' &\underset{(3.4)}{=} \int_\Omega (C : \varepsilon(q)) : \varepsilon(u') - \int_{\Gamma_0} (\sigma(q) \cdot n) \cdot u' \\ &\underset{(3.6)}{=} \int_{\Gamma_1} v_n f \cdot q + v_n \kappa p \cdot q - v_n(\sigma_\tau(u) : \nabla_\Gamma q) + \int_{\Gamma_0} (\sigma(q) \cdot n) v_n \frac{\partial u}{\partial n}\end{aligned}$$

Adding in the volume term $v_n j(u)$, we obtain the boundary shape gradient

$$D\mathscr{J}(\Omega; v) = \int_{\Gamma_1} v_n(j(u) + f \cdot q + \kappa p \cdot q - \sigma_\tau(u) : \nabla_\Gamma q) + \int_{\Gamma_0} v_n \frac{\partial u}{\partial n}(\sigma(q) \cdot n). \qquad (3.8)$$

4. The Strip Method for Shape Sensitivity

As described in Section 3, we encountered some difficulties in implementing the boundary method. The lower accuracy can cause optimization algorithms to converge significantly more slowly, and to worse answers. Quantities such as surface stress and strain that one might assume would be readily available in a structural dynamics code were not. The analytical work required to derive the boundary method expression 3.8 is somewhat involved, and would need to be repeated for different physics and boundary conditions. And yet, the volume method still suffers from the problem of being invasive to the FEM code: the details are shown in Section 2.1, where we see that the derivatives of the FEM matrix entries require access to the same routines that are used to assemble the FEM matrices themselves. We also have not yet answered the questions posed at the end of Section 1, namely what to do with interior gradient values computed by the volume method, and how to update the mesh during shape optimization.

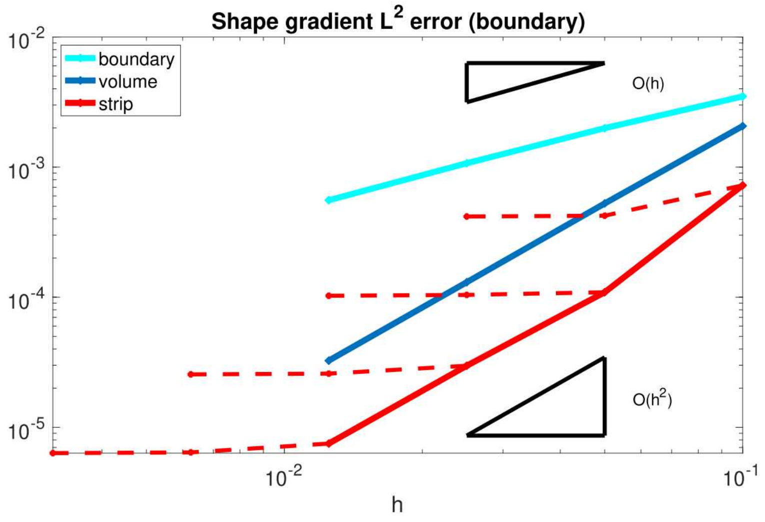

Figure 5. **L^2 error in the shape gradient with mesh refinement. Since we are evaluating the improvements that can be made relative to a given discretization, all refined solutions are projected back to the coarsest mesh prior to error computation. The "exact" solution is computed on a much finer mesh. In blue: the volume method as its single mesh is uniformly refined. In red: the strip method, where each dashed curve indicates the sequence of mesh refinements shown in Figure 6. In cyan: the boundary method.**

All of these considerations motivated the development of the *strip method* for shape derivatives. Our paper has been submitted to *Computer Methods in Applied Mechanics and Engineering*, and is available as a preprint [9]. The Hadamard structure theorem [5, Ch. 9, Th. 3.6] tells us that shape gradients are fully supported on the boundary of the domain. In our paper, we show that the volume method gradient values computed on the interior of the domain tend towards zero as the mesh is refined, and that they are at best unhelpful, and at worst detrimental to updating the mesh as the domain boundary is moved. Therefore, we can restrict our computations to a narrow strip (usually just one element wide) at the domain boundary. The real advantage of the strip method, however, comes not just from throwing away the interior values; the primary cost in gradient computation is typically the computation of the state and adjoint solutions rather than evaluation of the gradient itself. What is important is rather the realization that the strip method need not be implemented on the same mesh, nor even in the same code. Like the boundary method, all that is required is the ability to query the original simulation code for the state and adjoint solutions and their spatial gradients evaluated at quadrature points. This flexibility provides other advantages: the mesh used for gradient computation can be independently refined (the example shown in Figure 5 uses bilinear quadrilateral elements for the state and adjoint computations, and linear triangles for the shape gradient), allowing us to match the accuracy of a higher-order code for the state and adjoint without having to implement the shape calculus for higher-order elements. The strip method code could even implement the shape calculus using Automatic Differentiation, avoiding the need to go through the work described in Section 2.1.

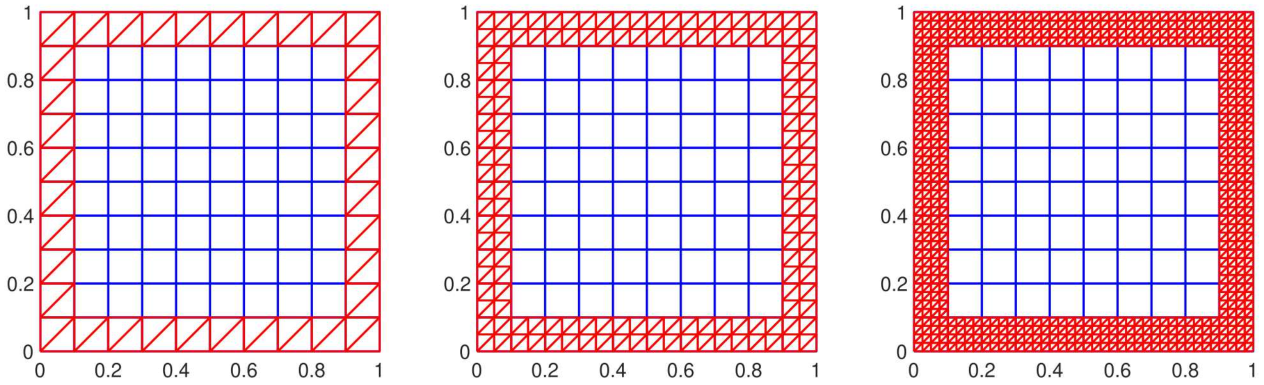

Figure 6. From left to right: successive levels of refinement of the strip mesh.

In the paper [9], we also demonstrate that error estimates developed in [10] can be extended to the strip method, i.e., that we retain the accuracy of the volume method.

5. Regularization of Surfaces

While more general classes of shape constraints must be considered in order to make a robust shape optimization capability that can handle design and manufacturing constraints, it seems that a reasonable starting point is to be able to maintain smoothness of the surfaces produced by gradient-based optimization algorithms, where the gradient is computed on the underlying simulation mesh. Most design problems of immediate practical interest could be attacked with a parametrization approach. However, this turns out not to be so simple. Aside from the simplest cases, it is impractical to parametrize meshes after the fact. Instead, what is needed to make this a workable strategy is a differentiable mapping from CAD geometry or some other geometric description to the mesh. In effect, this would require low-level interfaces between the CAD and meshing software, the optimization algorithm, the simulation code, and the shape gradient code. In other words, it requires close collaboration of at least four different teams in different organizations, which sounds unrealistic absent a major effort. However, some nice results have already been achieved using parametrization on problems of interest in specialized cases. These efforts are described in Section 5.1.

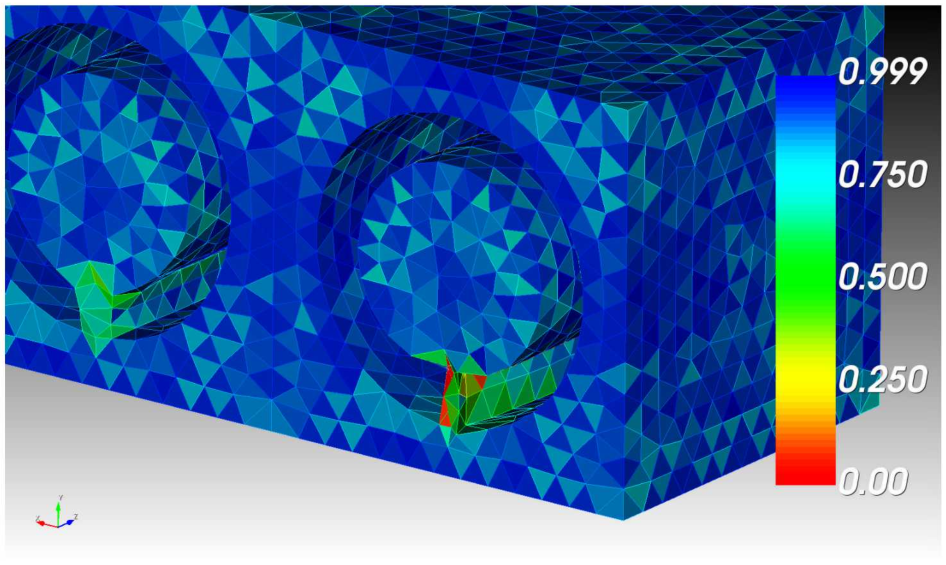

Figure 7. Shape optimization on a split-ring resonator without smoothness constraints results in localized, but severe, reductions in mesh quality. Values closer to 1 indicate better mesh quality.

Another approach that must at least be considered is to regularize or constrain some notion of the smoothness of the surfaces, treating the coordinates of each surface node as optimization variables. We've done initial investigations using estimates of the surface curvature computed on triangular surface meshes, but didn't have time to try it on realistic examples. The general ideas, as well as some interesting technical issues encountered are described in Section 5.2.

5.1. *Parametrized Surfaces*

Brett Clark and the Plato [20] team have implemented a nice demonstration capability for parametrized shape optimization that uses the shape calculus code developed in Sierra/SD by this project. Their purpose is to design mass mocks that, in addition to matching center of mass and moments of inertia, also match modal properties to some extent. The same machinery can be used for the problem of fixture design, where they have found that objective functions based on mode shapes and frequencies are better behaved than simple frequency response function (FRF) matching. The objective function that worked well was developed by Tyler Schoenherr [18]. The other key ingredients are the open-source code Engineering Sketch Pad (ESP) [8], which builds differentiable maps from CAD parameters to nodal mesh coordinates, Plato, which manages the problem description and communication between all of the moving parts, and the existing code in Sierra/SD that solves eigenvalue problems and provides a specialized linear solver for the singular adjoint system that arises.

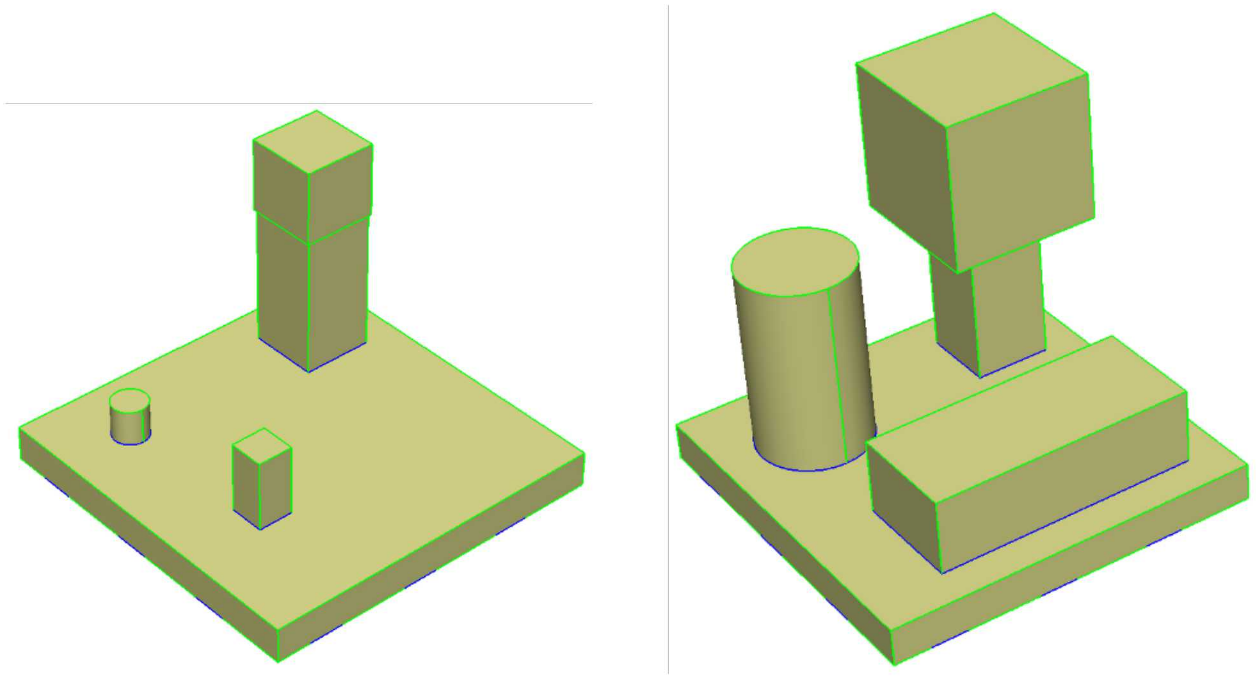

Figure 8. At left, the initial guess for the geometry of the fixture. Geometric parameters were length, height, radius, etc. At right, the target configuration on which the reference modal data was generated. The final result was very near to the original dimensions; see the data in Table 1.

ESP has some limitations: it handles only single blocks, generates only tetrahedral meshes, and doesn't work with existing CAD descriptions. Thus, it isn't practical to use this approach with a large existing CAD model or mesh. However, the results are quite good. The starting guess and target geometry are shown in Figure 8. After optimization, the results are visually indistinguishable from the target configuration. Numbers are given in Table 1.

We have also implemented a simple parametrized capability natively within Sierra/SD. It allows the association with a given mesh sideset of a perturbation in a particular Cartesian coordinate

Parameter	Before	After	Target
cylinder pos x	2	2.48	2.5
cylinder pos y	2	2.51	2.5
cylinder height	1	5.01	5
cylinder radius	0.5	1.51	1.5
weight width	2.1	3.92	4
weight height	2.1	4.11	4
weight depth	2.1	3.97	4
boss width	1	3.02	3
boss height	1	6.97	7

Table 1. Geometric parameters for the structure shown in Figure 8: before, after, and target values.

direction. This allows us, for example, to move around planar surfaces, which is already enough to provide a much nicer solution the problem shown in Figure 7. However, the inability to connect the CAD descriptions through the mesh to the optimization limits the utility of this approach.

5.2. *Surface Curvature*

The ability to make more general free-form changes to the surface shape is potentially very useful, particularly in vibration problems where seemingly subtle changes can result in dramatic changes in physical behavior, e.g., [1]. One obvious approach is to penalize or constrain mean surface curvature. Ryan Viertel led the work in comparing our implementation of the curvature estimation algorithm of Rusinkiewicz [16] to the results produced by fancier algorithms in libigl [12]: the Laplace-Beltrami, and quadric fitting methods. The Laplace-Beltrami method was on par with Rusinkiewicz's method but quadric fitting performed the best. However, the results of Rusinkiewicz's algorithm were reasonable, and its relative simplicity made implementation of shape derivatives (i.e., differential change in the surface curvature estimate with changes in surface coordinates) a tractable task. Based on these efforts, the ability to estimate both mean and Gaußian surface curvature and their shape derivatives was added to the shape optimization capabilities in Sierra/SD as an optional penalty term. There was not enough funding left to try it out in earnest on real problems, but it is thoroughly unit tested. Some experiments have been done looking at the integral of these curvature quantities vs sums over nodes. The integral formulations were found to be better-behaved in the sense of giving a gradient that is zero for a planar surface and that points perturbed planar surfaces back toward being planar.

Rusinkiewicz's algorithm estimates a "normal" vector at each mesh vertex, based on a Voronoi-area-weighted average of the neighboring triangle normal vectors. Thus, it is essential to be able to compute the Voronoi area in each triangle associated with a given vertex, and to compute its derivatives with respect to the Cartesian coordinates of the triangle's vertices. Computation of the Voronoi area proceeds differently in the cases of acute, right, and obtuse triangles. In the acute case, the circumcenter is inside the triangle, and the interior of the triangle can be divided into six triangles formed from the vertices, midpoints, and circumcenter. For a right triangle, the circumcenter lies on the hypotenuse, and the interior can be divided into four

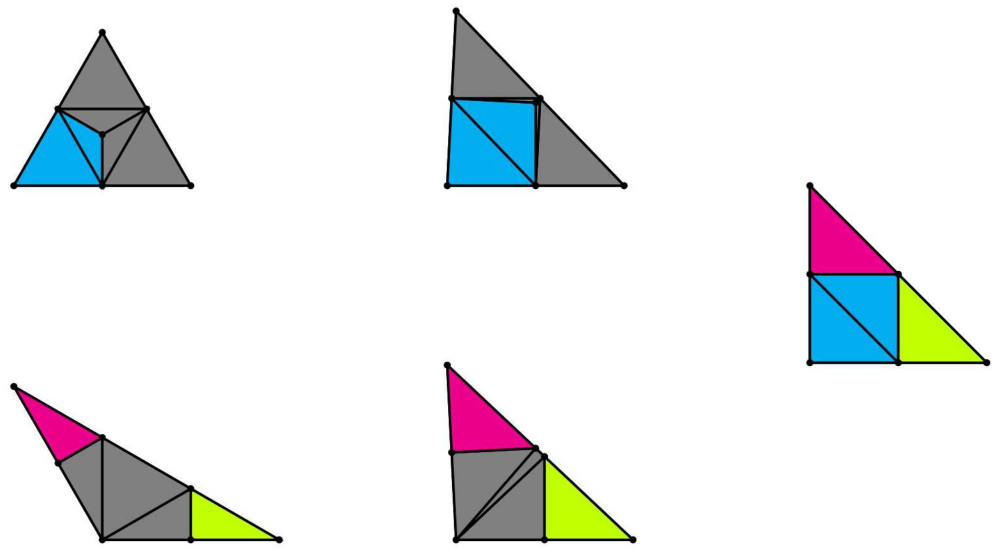

Figure 9. Constructions for Voronoi area: collapsing the acute and obtuse cases into the right case. The gradient computation in the right case is done using the colored triangles.

triangles formed from the vertices, midpoints, and circumcenter. For an obtuse triangle, the circumcenter lies outside the triangle, and the interior can be divided into five triangles formed from the vertices, two midpoints, and the points of intersection of their perpendicular bisectors with the opposite side of the triangle. See the illustrations in Figure 9. Shape calculus for the Voronoi area is tricky for a right triangle; the acute formulation is used for the right vertex, and the obtuse formulation is used for the other two vertices. This results in continuity of the triangles used to compute the associated Voronoi area in each case as the triangle passes through right to either acute or obtuse.

6. Conclusions

The main impact of this project is that we have developed knowledge of how to choose problem-appropriate shape derivative algorithms, laying the groundwork for a shape optimization capability that can meaningfully improve existing designs. We still wouldn't argue that any one algorithm for shape sensitivity is always superior, and rather we now have a toolkit that can be brought to bear on a variety of problem types. However, the new strip method is likely to be the best option in most cases, combining the accuracy of the volume method with the non-invasiveness into the code of the boundary method. Our paper, *The Strip Method for Shape Derivatives* has been submitted to *Computer Methods in Applied Mechanics and Engineering*, and is available as a preprint [9]. Work on this paper fostered an ongoing collaboration with Prof. Harbir Antil at George Mason University.

Through collaboration with ongoing projects, a significant set of new capabilities has been developed under the umbrella of this project:

- Shape calculus in Sierra/SD [4] for `directfrf`, `eigen`, and `transient` solution cases
- Implementation of Tyler Schoenherr's Modal Projection Error objective [18] in Sierra/SD
- Support of Plato [20] mass mock, fixture design capabilities; interface to Engineering Sketch Pad (ESP) [8]
- Surface Curvature code for regularization and constraints in shape optimization problems
- Matlab-based code and examples used for research paper

There has already been significant mission impact via the development of shape optimization methods to be used for fixture design in normal mechanical environments testing, i.e., helping ground-based tests to replicate real environments. Additionally, these capabilities are expected to be useful for vibration control applications. In the future, we would like to leverage the shape sensitivity capabilities for verification and validation purposes: they make possible characterization of the uncertainty associated with variations in manufactured shapes, and open the possibility of working with new tools being developed for optimization under uncertainty to produce robust designs.

REFERENCES

[1] Erik Bängtsson, Daniel Noreland, and Martin Berggren. Shape optimization of an acoustic horn. *Computer methods in applied mechanics and engineering*, 192(11-12):1533–1571, 2003.

[2] Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013.

[3] Martin Berggren. A unified discrete–continuous sensitivity analysis method for shape optimization. In *Applied and numerical partial differential equations*, pages 25–39. Springer, 2010.

[4] Gregory Bunting, Nathan K Crane, David M Day, Clark R Dohrmann, Brian Anthony Ferri, Robert C Flicek, Sean Hardesty, Payton Lindsay, Scott T Miller, Lynn Brendon Munday, et al. Sierra structural dynamics-users notes 4.50. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2018.

[5] M. C. Delfour and J.-P. Zolésio. *Shapes and geometries*, volume 22 of *Advances in Design and Control*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2011. Metrics, analysis, differential calculus, and optimization.

[6] L. Furbatto, G. Di Lorenzo, and C. Pedersen. Optimization in the abaqus environment using tosca, 2009.

[7] R.A. Gellatly and D.M. Dupree. Examples of computer-aided optimal design of structures. In *Proceedings of the 10th IABSE Congress*, 1976.

[8] Robert Haimes and John Dannenhoffer. The engineering sketch pad: A solid-modeling, feature-based, web-enabled system for building parametric geometry. In *21st AIAA Computational Fluid Dynamics Conference*, page 3073, 2013.

[9] S. Hardesty, H. Antil, D. Kouri, and D. Ridzal. The strip method for shape derivatives. Preprint available at `http://dx.doi.org/10.13140/RG.2.2.32766.82246`.

[10] Ralf Hiptmair, Alberto Paganini, and Sahar Sargheini. Comparison of approximate shape gradients. *BIT Numerical Mathematics*, 55(2):459–485, 2015.

[11] Daniel Alejandro Ibanez. *Conformal mesh adaptation on heterogeneous supercomputers*. PhD thesis, Rensselaer Polytechnic Institute, 2016.

[12] Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/.

[13] R Meske, J Sauter, and E Schnack. Nonparametric gradient-less shape optimization for real-world applications. *Structural and Multidisciplinary Optimization*, 30(3):201–218, 2005.

[14] Sebastian A Nørgaard, Max Sagebaum, Nicolas R Gauger, and Boyan S Lazarov. Applications of automatic differentiation in topology optimization. *Structural and Multidisciplinary Optimization*, 56(5):1135–1146, 2017.

[15] O. Pironneau. *Optimal shape design for elliptic systems*. Springer Series in Computational Physics. Springer-Verlag, New York, 1984.

[16] Szymon Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004.*, pages 486–493. IEEE, 2004.

[17] Stephan Schmidt. Weak and strong form shape hessians and their automatic generation. *SIAM Journal on Scientific Computing*, 40(2):C210–C233, 2018.

[18] Tyler Franklin Schoenherr, Jerry W. Rouse, and Julie Harvie. Characterizing dynamic test fixtures through the modal projection error. Technical Report SAND2020-1119, Sandia National Laboratories, 2020.

[19] J. Sokołowski and J.-P. Zolésio. *Introduction to shape optimization*, volume 16 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1992. Shape sensitivity analysis.

[20] Plato Development Team. Plato 2.3 user's manual. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2019.

[21] Sierra Toolkit Development Team. Sierra toolkit manual version 4.50. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2018.

This page intentionally left blank.

DISTRIBUTION

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
1	Sean Hardesty	1542	0845

Email—Internal (encrypt for OUO)

Name	Org.	Sandia Email Address
Technical Library	01177	libref@sandia.gov

This page intentionally left blank.