Using Design-Parameter Sensitivities in Adjoint-Based Design Environments

John F. Dannenhoffer, III*

Aerospace Computational Methods Laboratory

Syracuse University, Syracuse, NY, 13244

Robert Haimes†

Aerospace Computational Design Laboratory

Massachusetts Institute of Technology, Cambridge, MA, 02139

Over the past several years, considerable progress has been made in aerodynamic design through the use of adjoint-based solution technologies. These design systems allow one to change the surfaces of a configuration so that some objective function, such as lift-to-drag ratio or sonic boom strength, is optimized. Unfortunately, these systems change the configuration surfaces on a point-by-point basis, instead of by changing the design parameters that were used to generate the original configuration; this limitation arose from the lack of good sensitivity calculations through the geometric design process. The objective of this paper is to demonstrate the coupling of recently developed configuration sensitivity calculations with the adjoint-based optimization frameworks. In particular, a wing is optimized to minimize the induced drag (for a fixed lift) through both the `CART3D` and `FUN3D` design frameworks. Several methods for propagating sensitivity information into the interior of Faces were investigated. The optimized results, both for sensitivities computed by finite differences (which are nearly identical to the predicted displacement field but expensive to compute) and for sensitivities computed analytically (which disagree with the predicted displacements but are inexpensive to compute), are nearly identical.

I. Design Frameworks

Over the past several years, a few CFD solution systems have been extended to include a "design capability" through the use of adjoint-based flow solutions. Chief amongst these are `CART3D`,[1] `FUN3D`,[2] and `SU2`.[3] All of these systems provide the ability to optimize some user-defined objective function by changing the shape of the configuration based on parametrization schemes that are not necessarily consistent with the user's original parametrization. In order to get back to the user's design parameters, these frameworks allow a user to provide a "Jacobian" matrix that couples changes in the user's design parameters (such as wing sweep, taper ratio, and incidence distribution) with changes of the configuration shape.

Methods for generating these "Jacobian" matrices, such as `BandAids`[4] and `MASSOUD`[5] have been constructed without any knowledge of the process used to create the configuration in the first place. As such, these techniques can at best provide a Jacobian with respect to an approximation of the parameters. Also, these techniques are only useful for relatively simple configurations, such as a wing and body.

Recently, a new feature-based, parametric solid modeler, `ESP`[6] has been developed that allows one to both generate a configuration based upon its design parameters as well as determine the sensitivity of the

*Associate Professor, Mechanical and Aerospace Engineering, AIAA Associate Fellow.

†Principal Research Engineer, Department of Aeronautics & Astronautics, AIAA Member.

configuration shape with respect to these design parameters. The objective of this paper is to demonstrate the use of ESP directly within the CART3D and FUN3D design frameworks.

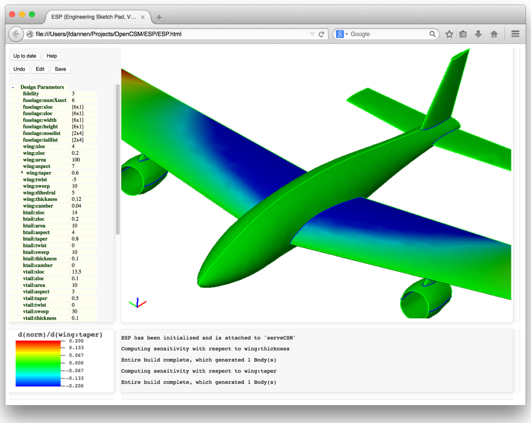
In the next section, background information is provided about ESP, with a special emphasis on the way in which it computes the sensitivity information that is needed to construct the Jacobian matrix. That is followed by sections that describe both the CART3D and FUN3D design frameworks. To demonstrate the use of ESP's parametric sensitivities, the shape of an inviscid wing is optimized to minimize the (induced) drag coefficient, while holding the wing's lift coefficient fixed. Results for both CART3D and FUN3D are shown.

II. The Engineering Sketch Pad (ESP)

The "Engineering Sketch Pad" (ESP) is an open-source, feature-based, parametric solid modeler.[6] As opposed to most commercially-available geometry-generating CAD systems, ESP was designed for geometries encountered in aerospace applications for which CFD (as well as other discipline's) computations are desired. It supports a full range of standard construction primitives (features), but also gives organizations the ability to attach their own user-defined primitives (UDPs) and components (UDCs). The description of an ESP model is in an ASCII (human-readable file) and has a simple API, making it easy to integrate ESP into other environments.

A unique and key feature of ESP is its ability to compute "geometric" and "tessellation" sensitivities analytically (as opposed to the requirement to use finite differences) for a wide variety of configurations.[7] It should be noted that there exists a finite-difference mechanism within ESP in order to support features that have not yet been differentiated or for testing the analytic calculations. This contains a general point tracking scheme that does not have the limiting assumptions of previous methods.[7]

"Geometric" sensitivities are used to answer the question: how does the local surface shape/location change as one or more of the design parameters are changed? So for a cylinder, the "geometric" sensitivities on the cylindrical walls of a cylinder are zero when the length of the cylinder is changed. Hence, "geometric" sensitivities can be discontinuous at hard Edges. Sample "geometric" sensitives are shown in Figure 1. Note the sharp and discontinuous nature of the sensitivity contours at the wing/fuselage junction.



Design Parameters	
fidelity	3
fuselage:numXsect	6
fuselage:xloc	[6x1]
fuselage:zloc	[6x1]
fuselage:width	[6x1]
fuselage:height	[6x1]
fuselage:noselist	[2x4]
fuselage:taillist	[2x4]
wing:xloc	4
wing:zloc	0.2
wing:area	100
wing:aspect	7
^ wing:taper	0.6
wing:twist	-5
wing:sweep	10
wing:dihedral	5
wing:thickness	0.12
wing:camber	0.04
htail:xloc	14
htail:zloc	0.2
htail:area	10
htail:aspect	4
htail:taper	0.8
htail:twist	0
htail:sweep	10
htail:thickness	0.1
htail:camber	0
vtail:xloc	13.5
vtail:zloc	0.1
vtail:area	10
vtail:aspect	3
vtail:taper	0.5
vtail:twist	0
vtail:sweep	30
vtail:thickness	0.1

```
ESP has been initialized and is attached to 'serveCSM'
Computing sensitivity with respect to wing:thickness
Entire build complete, which generated 1 Body(s)
Computing sensitivity with respect to wing:taper
Entire build complete, which generated 1 Body(s)
```

Figure 1. Geometric sensitivity with respect to wing thickness at the root section.

"Tessellation" sensitivities are used to answer the question: how does a grid point on the surface move as one or more of the design parameters are changed? These derivatives are continuous across geometric entities and are *mostly* consistent with mesh-based finite differences. Note that the motion of the grid points are influenced by the underlying "geometric" sensitivities, but also by the changes in the trimming curves associated with the surface. In addition, the change in the location is also influenced by the grid generation scheme used to place the point; hence to be technically correct, the sensitivity should be found by differentiating the grid generation scheme itself.

A. Computation of Surface Tessellation Sensitivities

As mentioned above, the tessellation sensitivity is used to specify the motion of each tessellation point with respect to the design parameter. In general, it takes the form

$$\frac{d\vec{x}}{dP} = \frac{\partial \vec{x}}{\partial P} + \frac{\partial \vec{x}}{\partial \vec{u}} \frac{\partial \vec{u}}{\partial P}$$

where $\vec{x}$ refers to the coordinates of a point on the surface, $\vec{u}$ are the parametric coordinates of the points on the surface, and P is a design Parameter. The first term accounts for the (normal) motion of the surface on which $\vec{x}$ lies and the second accounts for the motion of each tessellation point along the surface, which is due to changes in the trimming curves of the Face.

The first term, $\partial \vec{x}/\partial P$, can be found directly by differentiating the functions that generated the surface associated with the point. For points that lie on an Edge (between two Faces – called "left" and "right"), the evaluation of the "left" Face gives $(\partial w/\partial P)_{\rm left} = \sqrt{(\partial x/\partial P)^2 + (\partial y/\partial P)^2 + (\partial z/\partial P)^2}_{\rm left}$, which is pointed in the $\vec{n}_{\rm left}$ direction; similar evaluation on the "right" Face gives $(\partial w/\partial P)_{\rm right}$ and $\vec{n}_{\rm right}$.

The key now is to find a Edge sensitivity vector, $(\partial \vec{x}/\partial P)_{\rm edge}$ that is consistent with the two Face sensitivities and which is locally normal to the Edge. This is obtained by solving the matrix equation:

$$\begin{bmatrix} n_{x,\rm left} & n_{y,\rm left} & n_{z,\rm left} \\ n_{x,\rm right} & n_{y,\rm right} & n_{z,\rm right} \\ t_{x,\rm edge} & t_{y,\rm edge} & t_{z,\rm edge} \end{bmatrix} \begin{bmatrix} (\partial x/\partial P)_{\rm edge} \\ (\partial y/\partial P)_{\rm edge} \\ (\partial z/\partial P)_{\rm edge} \end{bmatrix} = \begin{bmatrix} (\partial w/\partial P)_{\rm left} \\ (\partial w/\partial P)_{\rm right} \\ 0 \end{bmatrix}$$

for $(\partial \vec{x}/\partial P)_{\rm edge}$. Here, $n_{x,\rm left}$ means the x component of $\vec{n}_{\rm left}$ and $t_{x,\rm edge}$ means the x component of the tangent vector along the Edge (that is, dx/dt along the Edge, where t is the Edge parametric coordinate).

As with the Face sensitivity, the Edge sensitivity only gives changes normal to the Edge; the component of the sensitivity along the Edge is automatically set to zero. The complete details of these computations are found in Dannenhoffer and Haimes.[7]

The difficult part of the above is determining how changes in the parametric coordinates, $\vec{u}$, on the boundaries propagate to changes of the tessellation points in the interior of the Face. In the sections that follow, three different techniques are described, where the first two can be considered surrogates for the last.

1. Mean Value Coordinates (MVC)

Mean value coordinates (MVC) were developed by Hormann and Floater[8] as a way of smoothly propagating from the boundary of a (possibly-multiply-connected) surface to its interior. Its formulation contains a generalization of barycentric coordinates to n-sided, 2-dimensional polygons.

Interpolation using MVC is a two-step process. In the first, weights (w_i) are computed for each of the boundary points for any given interior point (u_j, v_j). This computation is solely a function of the boundary points, and is given by

$$w_i = 2\frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{r_i}$$

where α_i is the angle between lines from interior point j to boundary points i and $i+1$ and r_i is the distance from interior point j to boundary point i. Hence, the weight associated with boundary point i is based solely on the location of the interior point (u_j, v_j) and the boundary points (u_{i-1}, v_{i-1}), (u_i, v_i) and (u_{i+1}, v_{i+1}). Notice that if the three boundary points ($i-1$, i, and $i+1$) are not traversed in a counterclockwise manner (when viewed from point j), the weight w_i may be negative. Once all the weights are computed, they are normalized so that their sum is unity.

In the second step, the value of any quantity in the interior (such as the sensitivity of $\vec{x}$ with respect to a design parameter) is computed as the linear combination of the sensitivities along the boundary, or

$$\left(\frac{\partial \vec{x}}{\partial P}\right)_j = \sum_{i \in \text{bnd}} w_i \left(\frac{\partial \vec{x}}{\partial P}\right)_i$$

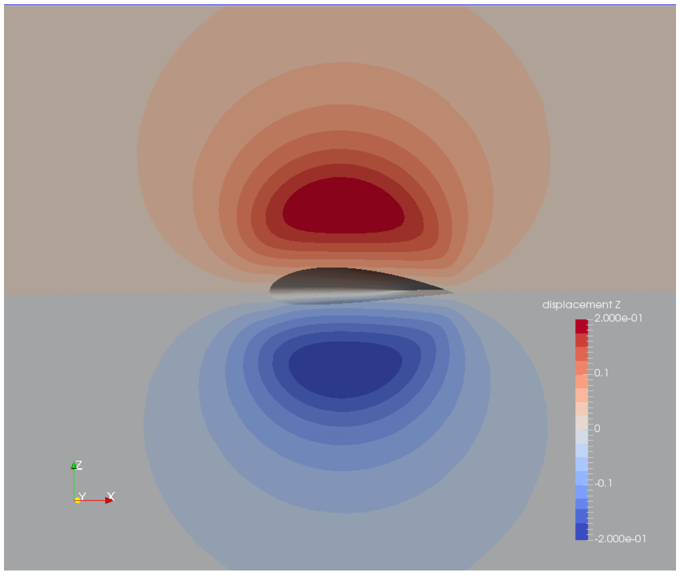


Figure 2. Contours of vertical displacement on the plane of symmetry, as computed by the original MVC.

As proved by Floater, MVC interpolation is very smooth, and is C^∞ everywhere. But, MVC only guarantees that the w_i are positive inside convex polygons. This can cause problems, as discussed by Lipman, *et al.*[9] In particular, consider the sensitivity of the vertical point motion with respect to airfoil thickness (on the symmetry plane), as shown in Fig. 2. As can be seen, the upper surface of the airfoil moves up (has a positive motion — shown in red) and the lower surface of the airfoil moves down (has a negative motion — shown in blue). Of particular concern, however, are the extrema that are located in the interior of the domain, as shown by red or blue "islands" away from the airfoil. These are due to the fact that all boundary points are used in the interpolation of each interior point. For interior points above the airfoil, both the positive motion of the upper surface and the negative motion of the lower surface contribute to the interpolation. Near the upper surface, the contribution of the upper surface is much stronger; but about a chord away, both surfaces have approximately the same contribution and hence the local extrema shown in Fig. 2. The effect of these local extrema are described in the results section.

2. Visibility-restricted MVC (VR-MVC)

To eliminate these local extrema, a change to MVC was made based upon a visibility argument. As noted above, MVC is known to have all positive weights, w_i, for convex domains. So the idea here is to use only the part of the domain that is convex with respect to the interior point (u_j, v_j) in the MVC calculation. To accomplish this, the MVC algorithm was modified so that only those boundary points that were visible to the interior point are included in the MVC sum; visibility is automatically determined by checking if a line segment from the interior point to the boundary point in question intersects any segment of the boundary. The results of this new visibility-restricted MVC (VR-MVC), which are shown in Fig. 3, demonstrate that the local extrema have been removed.

Of course, one of the potential problems associated with VR-MVC is that the smoothness of the interpolation function can no longer be guaranteed, since boundary points become visible in a discontinuous

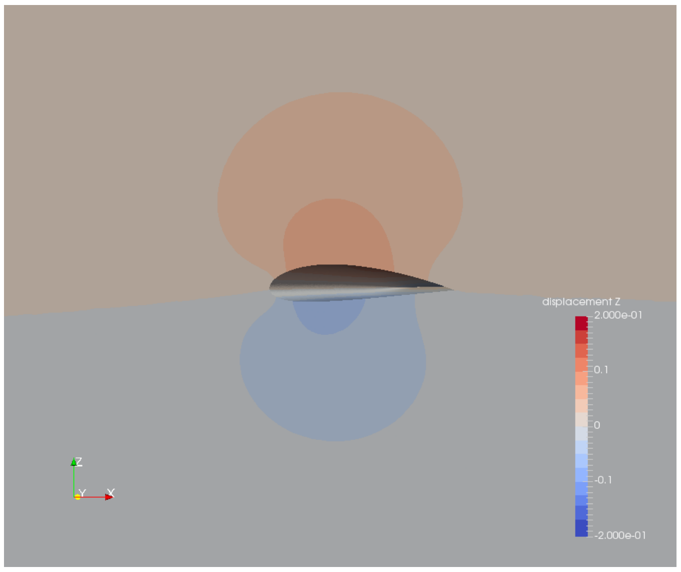


Figure 3. Contours of vertical displacement on the plane of symmetry, as computed by the visibility-restricted version of MVC.

manner. As seen in Fig. 3, in the *eye-ball norm*, the interpolating function appears smooth enough to be used for the current application, which will be demonstrated in the results section.

3. Differentiation of the Tessellator

Finally, one should be able to determine the motion of points in the interior of the discretized surface by differentiating the mesher that generated the original tessellation. In the current application, the Bowyer-Watson algorithm[10,11] is employed. It starts with an initial tessellation of a trapezoid that surrounds the boundaries. Then, each of the boundary points are successively added by removing any triangles whose circumcircle contain the new point; the star-shaped hole that is left is re-triangulated using the new point. After the boundary is tessellated, points are added in the interior for any triangle whose side length exceeds a specified tolerance. At the end, diagonal swapping is used to recover the triangle sides that conform to the original boundary.

At first glance, differentiating the above scheme seems daunting, due to the complexity of the operations used; for example, determination of which triangles to remove and which diagonals to swap involves significant computations and many logical decisions. But after careful examination, one can observe that the only part of the algorithm that really needs to be differentiated is the determination of the actual new point location (for the interior points). By leaving the testing and logic untouched, one can generate a new tessellation that is topologically equivalent to the original tessellation, even though it might not strictly satisfy the Delaunay criteria (note that having the same grid topology is more important here than a *correct* Delaunay mesh).

In the current implementation, all the interior points are placed to be at the centroid of the triangle whose side-length violates the side-length criterion. This is done with

$$\vec{u}_{\rm new} = (\vec{u}_0 + \vec{u}_1 + \vec{u}_2)/3$$

where $\vec{u}_0$, $\vec{u}_1$, and $\vec{u}_2$ are the parametric coordinates associated with (previously-placed) points in the tessellation. Differentiating the above (with respect to a design parameter, P) is straightforward, giving

$$\left(\frac{\partial \vec{u}}{\partial P}\right)_{\rm new} = \left[\left(\frac{\partial \vec{u}}{\partial P}\right)_0 + \left(\frac{\partial \vec{u}}{\partial P}\right)_1 + \left(\frac{\partial \vec{u}}{\partial P}\right)_2\right]/3$$

To compute the sensitivity of any surface grid point, one only needs to apply the above successively to each new tessellation point as it is generated. This can be accomplished by keeping track, for each point in the tessellation, the identity of the points that were used when the point was generated.

Results of using the differentiated tessellator are also shown in the results section.

III. CART3D Design Framework

A. Overall Process

The `CART3D` design framework solves constrained aerodynamic shape optimization problems. The framework coordinates the execution of the flow and adjoint solvers, and various utility codes to compute the value of the objective function, J, constraints, C, and gradients, $\mathrm{d}J/\mathrm{d}P$ and $\mathrm{d}C/\mathrm{d}P$, at each step of the optimization procedure for given design variables P. The framework uses multilevel parallelism to minimize design-cycle time and is designed to interface with virtually any optimization package. In this work, we use the SNOPT[12] optimizer. The details of the framework can be found in Nemec and Aftosmis,[1] here we provide only a brief summary. Recent examples of shape optimizations performed by this framework can be found in Refs. 15 and 16.

The framework controls the geometry build process through use of the Extensible Design Description Markup (XDDM) protocol.[1] This provides a modeler-neutral access to parameters and other information associated with the geometry construction and queries. It can be used with CAD-based and non-CAD modelers. This XML-based protocol is used throughout the design framework to express design variables, analysis parameters, objectives and constraints. It also provides access to a collection of standard services including the computation of triangulation shape sensitivities and symbolic function manipulation.

The flow is modeled by the Euler equations, which are discretized on a Cartesian mesh with embedded boundaries. The mesh consists of regular Cartesian hexahedra everywhere, except for a layer of body-intersecting cells, or cut cells, adjacent to the boundaries, as illustrated in Fig. 4. The spatial discretization

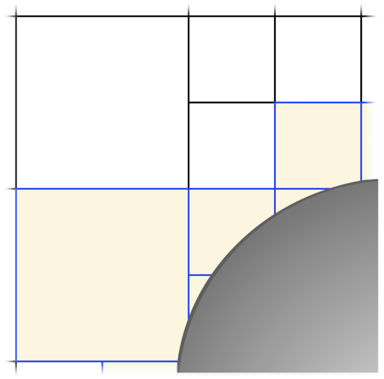


Figure 4. Multilevel Cartesian mesh in two-dimensions with a cut-cell boundary.

uses a cell-centered, second-order accurate finite volume method with a weak imposition of boundary conditions, resulting in a system of residual equations

$$\mathbf{R}(P, \mathbf{M}, \mathbf{Q}) = 0 \tag{1}$$

where $\mathbf{M}$ is the computational mesh and $\mathbf{Q}$ the flow solution vector. The design variables that appear directly in Eq. 1 involve parameters that do not change the computational domain, such as the Mach number, angle of attack, and side-slip angle. The influence of shape design variables on the residuals is implicit via the computational mesh

$$\mathbf{M} = f[\mathbf{T}(P)] \tag{2}$$

where $\mathbf{T}$ denotes a triangulation of the wetted surface.

The gradient is obtained by linearizing the objective function, $J(P, \mathbf{M}, \mathbf{Q})$, and the residual equations, resulting in the following expression

$$\frac{\mathrm{d}J}{\mathrm{d}P} = \frac{\partial J}{\partial P} + \frac{\partial J}{\partial \mathbf{M}}\frac{\partial \mathbf{M}}{\partial \mathbf{T}}\frac{\partial \mathbf{T}}{\partial P} - \psi^{\mathrm{T}}\left(\frac{\partial \mathbf{R}}{\partial P} + \frac{\partial \mathbf{R}}{\partial \mathbf{M}}\frac{\partial \mathbf{M}}{\partial \mathbf{T}}\frac{\partial \mathbf{T}}{\partial P}\right) \tag{3}$$

where ψ is an adjoint vector. A similar expression is obtained for each constraint. In embedded-boundary Cartesian methods, an infinitesimal perturbation of the boundary shape affects only the cut cells. The detailed evaluation of Eq. 3 is presented in Refs. 13,14. In the next section, we briefly outline the evaluation of the partial derivative term involving the sensitivities of the surface triangulation, $\partial\mathbf{T}/\partial P$. These are provided by ESP to the `CART3D` design framework, thereby circumventing the need to use finite-difference approximations.

B. ESPxddm

In order to link `ESP` with the `CART3D` design framework, a software module is required that can provide a closed and watertight triangulation of the body of interest at the current design (the suite of design parameters *at play* and their current values). If the software module can provide the design parameter sensitivities (Jacobian matrix, which contains the sensitivity of each surface tessellation point with respect to specified design parameter), then the `CART3D` design framework will use these sensitivities. If the parametric derivatives are not supplied, then the framework will finite-difference a perturbed body generated by this same software module.

The connection is made by the `ESP` application `ESPxddm`, which deals directly with Extensible Design Description Markup language. An XML file is constructed that describes which design parameters are active and what their start values are (as well as their valid range). This file also contains a pointer to the `.csm` file that describes the configuration. The triangulation is generated by the `EGADS` tessellator, which is an integral part of the `ESP` suite of software. The quality of the tessellation can be controlled via `ESP` attribution found in the `.csm` file and/or the data in the XML file.

As input, `ESPxddm` accepts the XDDM (XML) file, which describes the data that is being requested by the design framework. The following steps are then taken by `ESPxddm`:

1. Read the `.csm` file through the `OpenCSM` API.
2. Override the specified design parameter values with those found in the XDDM description.
3. Use the `OpenCSM` API to regenerate the model with the current state of the design.
4. Invoke the `EGADS` triangulator to produce the surface tessellation of the resultant geometry.
5. If requested, use the `OpenCSM` API to compute the analytic "tessellation" sensitivities for the specified design parameters at the triangulation points.
6. Write out this data in a `CART3D` *trix* file (which is basically a VTK unstructured data file).

The `ESPxddm` application is invoked by a script (`ESP.csh`), which is specified in the XDDM description, and is itself driven by a top-level optimization script.

`ESPxddm` has been available from within the `ESP` suite of software since Rev 1.08.

IV. FUN3D Design Framework

The `FUN3D` design framework consists primarily of a CFD flow solver and a volume mesh deformation solver based on linear elasticity. For gradient based optimization, sensitivity analysis is provided by the solution of the associated adjoint problems for the flow and mesh deformation. The process is described in detail by Nielsen,[17] and is summarized here subject to the specifics of the problem under consideration.

A. Flow Solver

For the current work, the flow solver solves the 3-Dimensional, steady, Euler equations given by

$$\frac{1}{V}\oint_{\partial\Omega} \mathbf{F}\cdot\hat{\mathbf{n}}\,dS = 0 \tag{4}$$

where $\hat{\mathbf{n}}$ is an outward-pointing unit normal, and V is the control volume bounded by the surface $\partial\Omega$. The inviscid flux tensor is given by

$$\mathbf{F} = \begin{bmatrix} \rho \\ \rho\mathbf{u}^T\mathbf{u} + p\mathbf{I} \\ \mathbf{u}\,(E+p) \end{bmatrix} \tag{5}$$

Here, ρ is the density, $\mathbf{u}$ is the absolute velocity vector, $\mathbf{u} = [u,v,w]^T$, E is total energy per unit volume, and $\mathbf{I}$ is the identity matrix. The equations are closed with the equation of state for a perfect gas.

$$p = (\gamma-1)\left[E - \rho\tfrac{\left(u^2+v^2+w^2\right)}{2}\right]. \tag{6}$$

The `FUN3D` flow solver[18–21] can be used to perform aerodynamic simulations across the speed range, and an extensive list of options and solution mechanisms is available for spatial and temporal discretizations on general static or dynamic mixed-element unstructured meshes that may or may not contain overset mesh topologies.

In the current work, the spatial discretization uses a finite-volume approach in which the dependent variables are stored at the vertices of single-block tetrahedral meshes. Inviscid fluxes at cell interfaces are computed using the upwind scheme of Roe.[22] Scalable parallelization is achieved through domain decomposition and message-passing communication.

B. Mesh Deformation

To deform the interior of the computational mesh as the surface mesh evolves during a shape-optimization procedure, the mesh is assumed to obey the linear elasticity equations of solid mechanics. These relations can be written as

$$\oint_{\partial\Omega} \lambda\left(\sum_{i=1}^{3}\frac{\partial u_i}{\partial x_i}\right)\mathbf{I}\cdot\hat{\mathbf{n}}\,dS + \oint_{\partial\Omega} 2\mu\boldsymbol{\varepsilon}\cdot\hat{\mathbf{n}}\,dS = 0 \tag{7}$$

where

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{8}$$

is the strain tensor, u_i is the displacement vector in each of the Cartesian coordinate directions, x_i, and λ and μ are material properties of the elastic medium. The quantities λ and μ are related to Young's modulus, E, and Poisson's ratio, ν, through the following:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} \tag{9}$$

and

$$\mu = \frac{E}{2(1+\nu)} \tag{10}$$

The system is closed with the specification of two of the four parameters λ, μ, E, and ν. In the current implementation, E is taken as inversely proportional to the distance from the nearest solid boundary, while Poisson's ratio is taken uniformly as zero. This approach forces all cells that are near boundaries to move in a nearly rigid fashion, while cells that are far from the boundaries are allowed to deform more freely. The system of equations is solved using GMRES[23] with either a point-implicit or ILU(0) preconditioning technique.[21,24]

C. Discrete Adjoint Equations

To derive the discrete adjoint equations, a compact notation is introduced for the governing equations that are outlined above. The spatial residual vector $\mathbf{R}$ of Eq. (4) is defined as

$$\mathbf{R} \equiv \oint_{\partial\Omega} \mathbf{F} \cdot \hat{\mathbf{n}}\, dS \tag{11}$$

Furthermore, the linear system of equations given by Eq. (7) can be written as

$$\mathbf{KX} = \mathbf{X}_{surf} \tag{12}$$

where $\mathbf{K}$ is the elasticity coefficient matrix that results from the discretization of Eq. (7), $\mathbf{X}$ is the vector of the mesh-point coordinates, and $\mathbf{X}_{surf}$ is the vector of known surface mesh-point coordinates, complemented by zeros for all interior coordinates. With the approach that was taken by Nielsen,[17] a Lagrangian function can be defined as follows:

$$L\left(\mathbf{P}, \mathbf{Q}, \mathbf{X}, \mathbf{\Lambda}_f, \mathbf{\Lambda}_m\right) = f\left(\mathbf{P}, \mathbf{Q}, \mathbf{X}\right) + \mathbf{\Lambda}_f^T \mathbf{R}\left(\mathbf{P}, \mathbf{Q}, \mathbf{X}\right) + \mathbf{\Lambda}_m^T\left(\mathbf{KX} - \mathbf{X}_{surf}\right) \tag{13}$$

where: $\mathbf{P}$ represents a vector of design variables; $\mathbf{Q}$ is the vector of volume-averaged conserved variables, $\mathbf{Q} = [\rho, \rho u, \rho v, \rho w, E]^T$; f is an objective function; and $\mathbf{\Lambda}_f$ and $\mathbf{\Lambda}_m$ are the adjoint variables that multiply the residuals of the flow and the mesh equations, respectively. In this manner, the governing equations may be viewed as constraints.

Differentiating Eq. (13) with respect to $\mathbf{P}$ and equating the $\partial\mathbf{Q}/\partial\mathbf{P}$ and $\partial\mathbf{X}/\partial\mathbf{P}$ coefficients to zero yields the discrete adjoint equations for the flowfield and mesh, respectively:

$$\left[\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right]^T \mathbf{\Lambda}_f = -\frac{\partial f}{\partial \mathbf{Q}} \tag{14}$$

and

$$\mathbf{K}^T \mathbf{\Lambda}_m = -\left\{\frac{\partial f}{\partial \mathbf{X}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right]^T \mathbf{\Lambda}_f\right\} \tag{15}$$

The remainder of the terms in the linearized Lagrangian can be grouped to form an expression for the final sensitivity vector:

$$\frac{dL}{d\mathbf{P}} = \frac{\partial f}{\partial \mathbf{P}} + \mathbf{\Lambda}_f^T \frac{\partial \mathbf{R}}{\partial \mathbf{P}} - \mathbf{\Lambda}_m^T \left[\frac{\partial \mathbf{X}_{surf}}{\partial \mathbf{P}}\right] \tag{16}$$

Equations (14) and (15) provide an efficient means for determining discretely consistent sensitivity information. The expense that is associated with solving these equations is independent of $\mathbf{P}$ and is similar to that of the governing equations. After the solutions for $\mathbf{\Lambda}_f$ and $\mathbf{\Lambda}_m$ have been determined, then the desired sensitivities may be calculated using Eq. (16), for which the computational cost is negligible.

A discrete adjoint implementation has been developed[17,20,24–26] for the flow solution method that is described above. The flowfield adjoint equations are solved in an exact dual manner, which guarantees an asymptotic convergence rate that is identical to the primal problem and costate variables that are discretely adjoint at every iteration of the solution process. The grid adjoint equations are solved using GMRES in a manner that is identical to the method used for Eq. (7).

D. Objectives and Constraints

The implementation of Nielsen[17] permits multiple objective functions, f_i, and explicit constraints, c_j, of the following form, each containing a summation of n_i and m_j individual components, respectively:

$$f_i = \sum_{k=1}^{n_i} \omega_k \left(C_k - C_k^*\right)^{p_k} \tag{17}$$

and

$$c_j = \sum_{k=1}^{m_j} \omega_k \left(C_k - C_k^*\right)^{p_k} \tag{18}$$

Here, ω_k represents a user-defined weighting factor, C_k is an aerodynamic coefficient such as total drag or the pressure or viscous contributions to such quantities, and p_k is a user-defined exponent. The $*$ superscript indicates a user-defined target value of C_k. Furthermore, the user may specify the boundaries in the grid to which each component function applies.

E. fun3dCSM

In order to link `FUN3D` and `ESP`, a new software module, named `fun3dCSM` was created. It takes as input the `.csm` file that describes the configuration and the current values for each of the design parameters, and produces both an updated surface tessellation and the Jacobian matrix, which contains the sensitivity of each surface tessellation point with respect to each design parameter.

As a prelude to using `fun3dCSM` in the design mode, it is necessary to generate the original configuration, and hence surface grid, that is to be used by `FUN3D`. This is done by running `EPS`'s `serveCSM` program, which performs the following operations:

1. Read a `.csm` file that describes the configuration. For the case below of an isolated wing, the wing is defined in terms of its area, aspect ratio, taper ratio, and leading edge sweep. In addition, each of the six cross-sections (which are evenly spaced between the root and tip) are NACA airfoils defined in terms of their thickness ratio, camber ratio, and incidence angle.

2. Build the configuration. This involves the generation of the six spanwise cross-sections that are blended into a smooth wing, which in turn is subtracted from a large box that represents the far-field. Also during this process, attributes are added to the resulting Boundary Representation (BRep) in order to specify suitable grid spacings.

3. Perform a nominal tessellation. This is done using `EGADS`'s internal tessellator, which first generates the points on each Edge and then generates the points in the interior of the Faces. This two-step process guarantees that the resulting tessellation is watertight.

4. Modify the Face tessellations, using an Delaunay-based surface tessellation. This process only modifies the locations of grid points inside Faces, and hence does not jeopardize the watertightness of the resulting grid.

5. Dump output files, which consist of the surface grids, the attributed BRep, and a file through which the resulting Delaunay-based surface tessellation can be morphed (as needed below).

The above operations are only executed once, before `FUN3D`'s design environment is executed.

During the execution of the `FUN3D` design environment, every time the system needs to generate a new configuration or needs sensitivity information, it calls `fun3dCSM`, whose process can be summarized as:

1. Load the `.csm` file that contains the description of the configuration, including the initial values for each of the design parameters. This is the same file as was used during step 1 of the initialization (as described above).

2. Build the initial configuration, using the same tessellation process as described above in steps 3 and 4 of the initialization process.

3. Update the design parameters, which are prescribed by the optimizer in `FUN3D`'s design environment.

4. Build the configuration, using the prescription in the `.csm` file, together with the latest design parameters.

5. Check that the topology of the new configuration (from step 4) is consistent with the topology of the original configuration (from step 2). These topological checks ensure that both BReps have the same number of Nodes, Edges, and Faces, and that they are connected in the same manner. If they are not consistent, then an error is raised that informs the user that the design changes are too big. Otherwise, a mapping is created between the identities of each component in the two BReps.

6. Create a tessellation for this new configuration that is topologically consistent with the tessellation created in step 2, but which conforms to the latest shape.

7. Compute the sensitivity at each surface grid point with respect to each design parameter. For this case, these sensitivities can done analytically because `ESP` has explicit analytical derivatives for the operations used to generate the current configuration.

8. Write a file that contains both the updated surface tessellation as well as all the sensitivities.

F. Treatment of Surface Sensitivities

The implementation the `FUN3D` framework is sufficiently general such that the user is able to employ a geometric parametrization scheme of choice, provided that the associated linearizations required by the adjoint method described above are also available. This is consistent with the implementation for the `CART3D` design framework.

V. Design Example

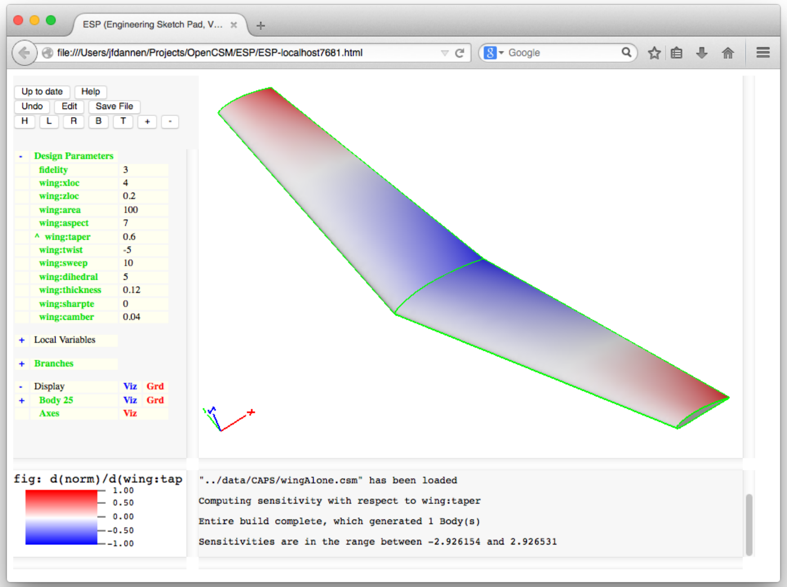
The sample design cases consist of a swept wing, whose design parameters are the thickness, camber, and incidence distributions each at 6 spanwise stations as can be seen in Fig. 5. For this design study, the wing's aspect ratio, taper ratio, leading edge sweep, and dihedral angles are held fixed. In addition, only one half of the wing is used. It was found constraints were needed to limit the overall changes in wing shape so as to not end up with grid tangling when the volume grids are deformed during `FUN3D`'s linear elasticity operation.

Design calculations are being done in both the `CART3D` and `FUN3D` design frameworks. For the `FUN3D` framework, only inviscid effects will be considered. Therefore, for both frameworks, the drag will simply be the induced drag.

A. Treatment of sensitivities in `FUN3D` environment

The major challenge in computing sensitivities that are consistent with changes in the tessellations from one design iteration to the next is the way in which the sensitivities are computed along the BRep Edges. When computing Edge "tessellation" sensitivities, one has to ensure that the shape of the new Edge is predicted consistently. But the spacing of points *along* the Edge is somewhat arbitrary. In the description of `ESP`'s tessellation sensitivities (above), it was noted that `ESP` returns only the part of the sensitivity that is normal to the Edge.

This results in the situation shown in Fig. 6, which depicts the Edges at the root of the wing (i.e., at the plane of symmetry). Here the solid line (with circles) represents the shape of the airfoil with the initial design parameters; the dashed line (with triangles) represents the airfoil shape after the airfoil's thickness has changed. The little lines (with circles) emanating from the airfoil show the predicted sensitivities. (The


ESP (Engineering Sketch Pad, V...
file:///Users/jfdannen/Projects/OpenCSM/ESP/ESP-localhost7681.html
Google
Up to date
Help
Undo
Edit
Save File
H
L
R
B
T
+
-
fig: d(norm)/d(wing:tap
1.00
0.50
0.00
−0.50
−1.00

-	Design Parameters	
	fidelity	3
	wing:xloc	4
	wing:zloc	0.2
	wing:area	100
	wing:aspect	7
	^ wing:taper	0.6
	wing:twist	-5
	wing:sweep	10
	wing:dihedral	5
	wing:thickness	0.12
	wing:sharpte	0
	wing:camber	0.04
+	Local Variables	
+	Branches	
-	Display	Viz Grd
+	Body 25	Viz Grd
	Axes	Viz

```
"../data/CAPS/wingAlone.csm" has been loaded
Computing sensitivity with respect to wing:taper
Entire build complete, which generated 1 Body(s)
Sensitivities are in the range between -2.926154 and 2.926531
```

Figure 5. Sample wing configuration for optimization, shown colored by geometric sensitivity with respect to overall wing twist.

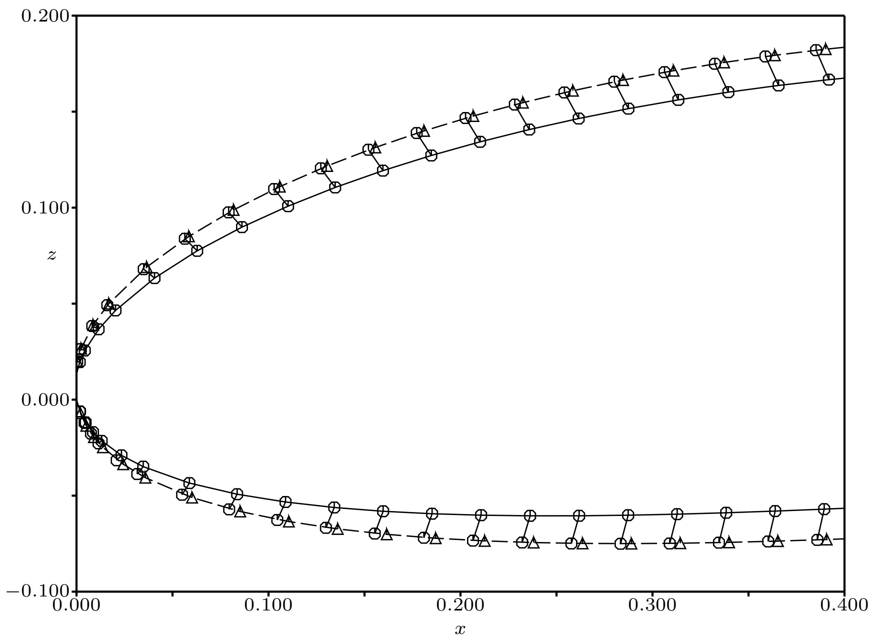


Figure 6. Airfoil shape on the symmetry plane. The solid line (with circles) shows the airfoil shape based upon the initial design parameters. The dashed line (with triangles) shows the airfoil shape after the thickness of the root section has been increased. The spines (with circles) show the predicted sensitivities.

Table 1. Description of the 7 `FUN3D` test cases.

Case number	sensitivity method	Displacement method
1	diff of tess	barycentric
2	MVC	barycentric
3	VR-MVC	barycentric
4	diff of tess	morphing
5	MVC	morphing
6	VR-MVC	morphing
7	finite diffs	morphing

actual displacements and sensitivities have been exaggerated by several orders of magnitude in the figure for clarity.) Note that the outer circles do not coincide with the triangles, but that all the circles lie on the dashed line. This shows that `ESP`'s sensitivity calculation correctly predicts the motion of the surface, but not necessarily the spacing of points on the new surface.

In order to assess the importance of this discrepancy, seven test cases were developed for the `FUN3D` design environment, as summarized in Table 1. "Sensitivity method" refers to the way in which sensitivities along the Edges are propagated to the interior of the Faces:

diff of tess uses the differentiation of the tessellator

MVC uses the original mean-value coordinates

VR-MVC use the visibility-restricted mean-value coordinates

finite diffs uses finite differences between tessellations generated for the original and perturbed configurations

"Displacement method" refers to the way in which the tessellation points are placed in the interior of the Faces:

barycentric uses `EGADS`' built-in method, which employs a triangle-based barycentric interpolation of a triangulation that only involves the BRep Face's boundary

morphing uses a morph of the Delaunay tessellation used on the original configuration

Figs. 7 and 8 show the evolution of the lift-to-drag ratio and cumulative number of flow solver iterations, both as a function of the number of flow solver (and adjoint solver) calls, for the `FUN3D` design environment.

Figures 9 through 15 show contours of the sensitivity and displacements on the plane of symmetry for a change in the thickness of the airfoil section at the root. The color contours represent the displacements, where the "Displacement method" is as described in Table 1. The line contours show the analytically-computed sensitivities, which correspond to the "Sensitivity method" from Table 1. Note the discrepancies in the contours near the wing surface, which is due to the "sliding" effect shown in Fig. 6. The contours only match for case 7, in which the sensitivities were computed by finite-differencing the tessellations. Note that these finite differences are very expensive to compute, since a complete new configuration and tessellation is required for each of the 18 active design parameters.

In order to assess the importance of the discrepancies shown in Figures 9 through 14, all cases were executed in the `FUN3D` design environment, with the results shown in Figures 7 and 8. Note that the final optimized results are nearly identical and that the number of required flow solver iterations are also nearly the same, with case 7 showing a very slight reduction in the required number of flow solver iterations. Based

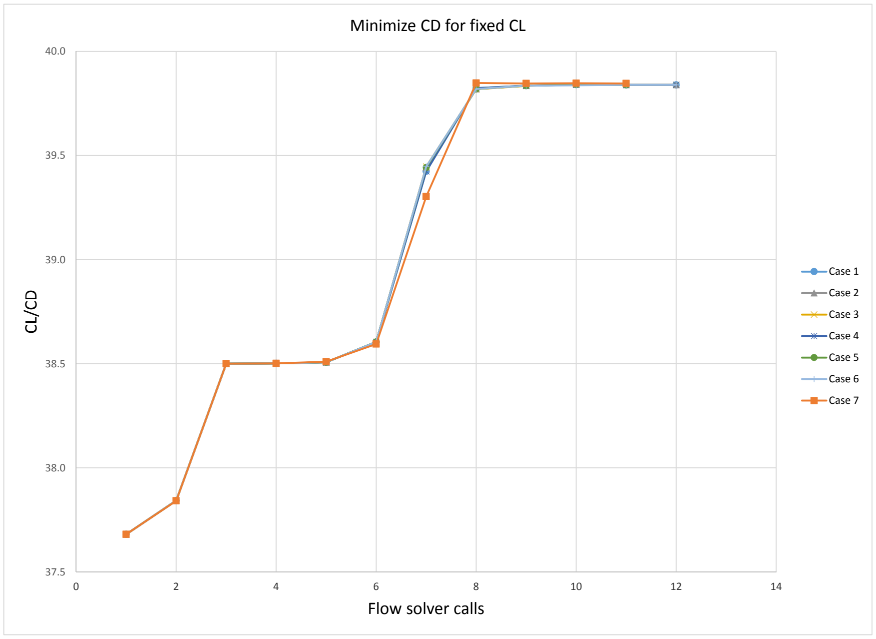


Figure 7. C_L/C_D improvement for a fixed lift with the `FUN3D` design environment for Cases #1 through #7.

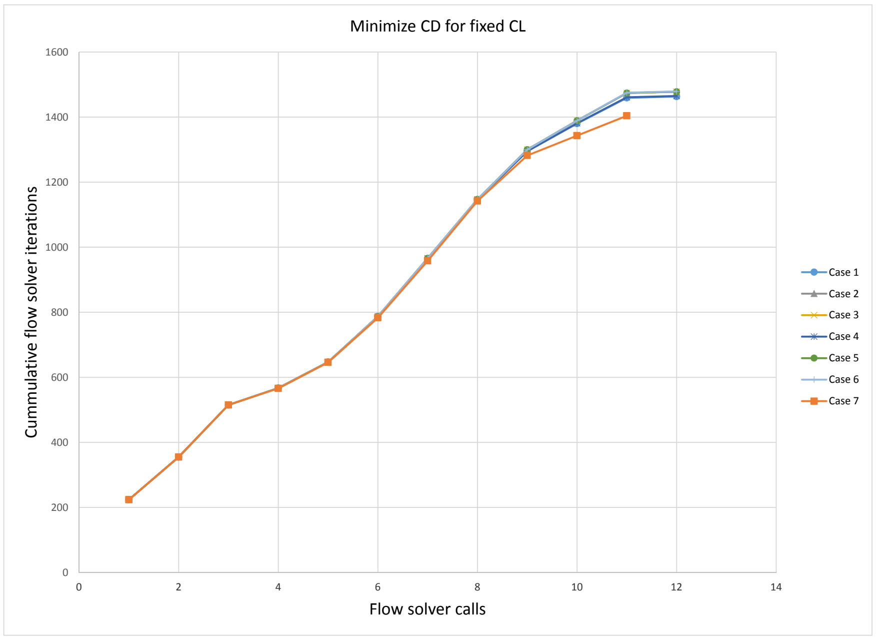
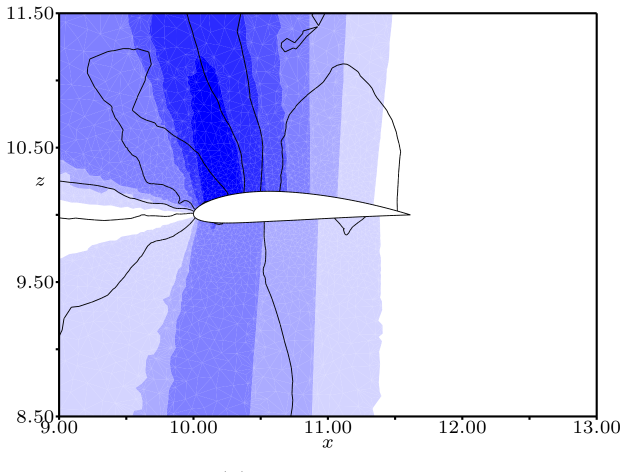
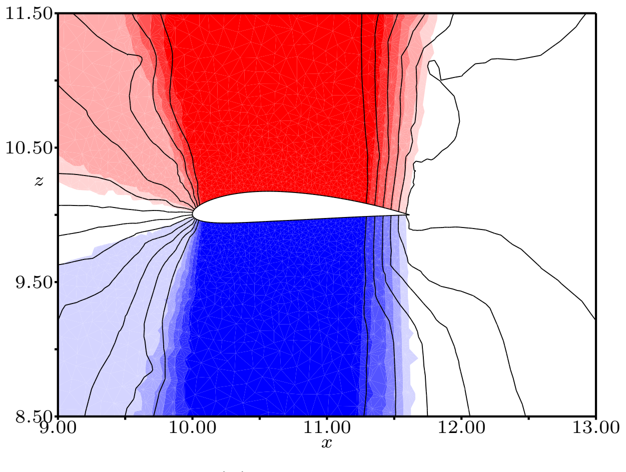


Figure 8. Number of flow solver iterations for a fixed lift with the `FUN3D` design environment for Cases #1 through #7.

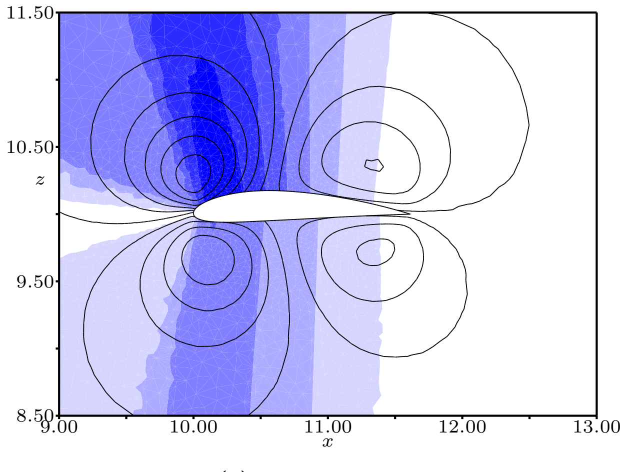


(a) x-component

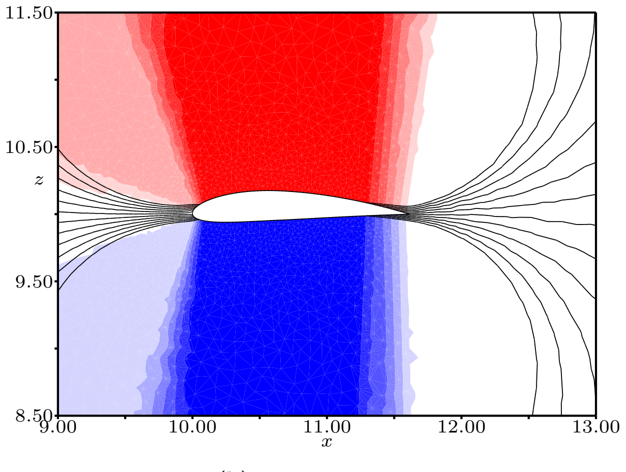


(b) z-component

Figure 9. Comparison of sensitivity and displacements for Case #1 on the symmetry plane due to changes in the thickness at the wing root. Color contours are displacements between baseline and perturbed configuration. Line are contours of analytically-computed sensitivity.

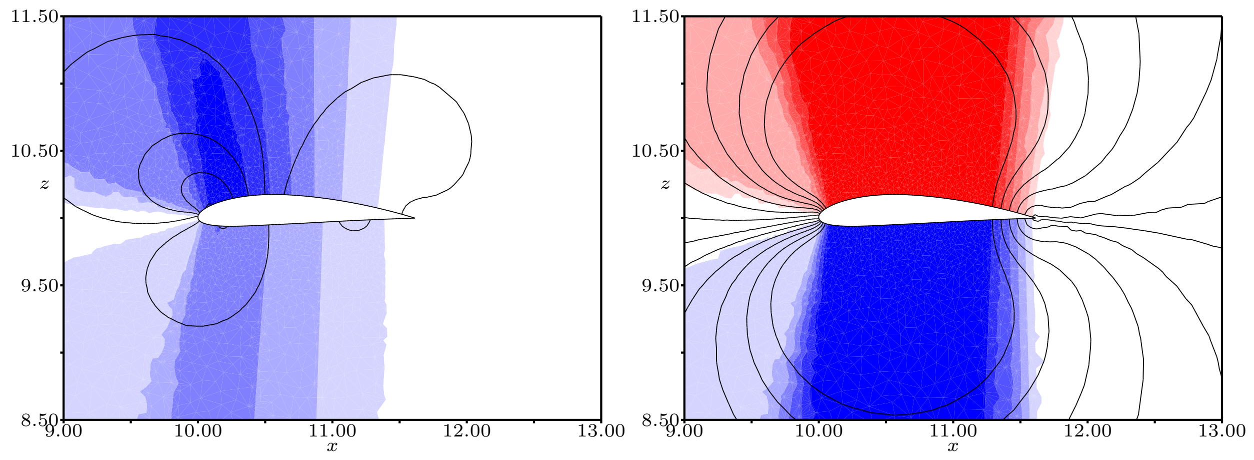


(a) x-component



(b) z-component

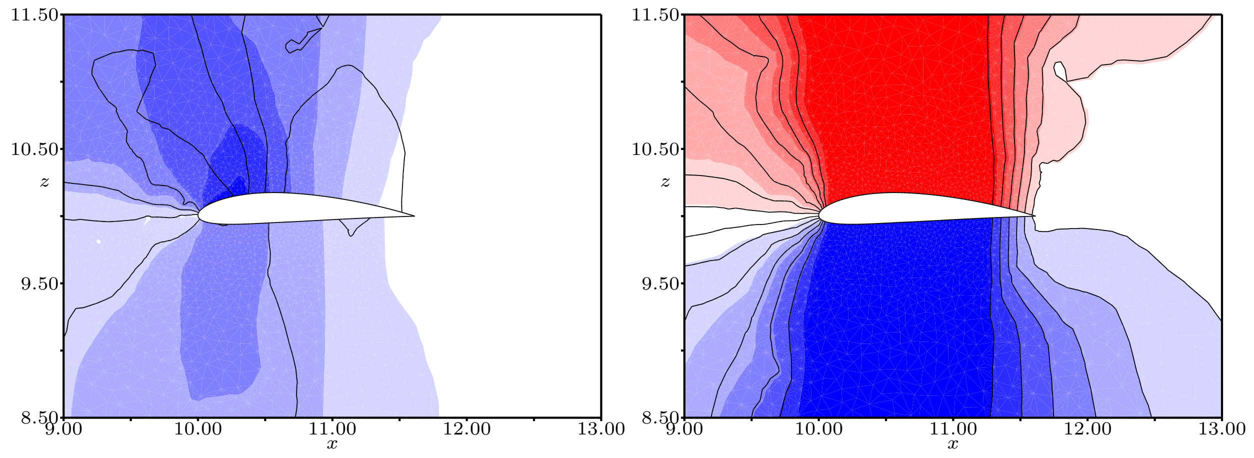
Figure 10. Comparison of sensitivity and displacements for Case #2 on the symmetry plane due to changes in the thickness at the wing root. Color contours are displacements between baseline and perturbed configuration. Line are contours of analytically-computed sensitivity.



(a) x-component

(b) z-component

Figure 11. Comparison of sensitivity and displacements for Case #3 on the symmetry plane due to changes in the thickness at the wing root. Color contours are displacements between baseline and perturbed configuration. Line are contours of analytically-computed sensitivity.



(a) x-component

(b) z-component

Figure 12. Comparison of sensitivity and displacements for Case #4 on the symmetry plane due to changes in the thickness at the wing root. Color contours are displacements between baseline and perturbed configuration. Line are contours of analytically-computed sensitivity.

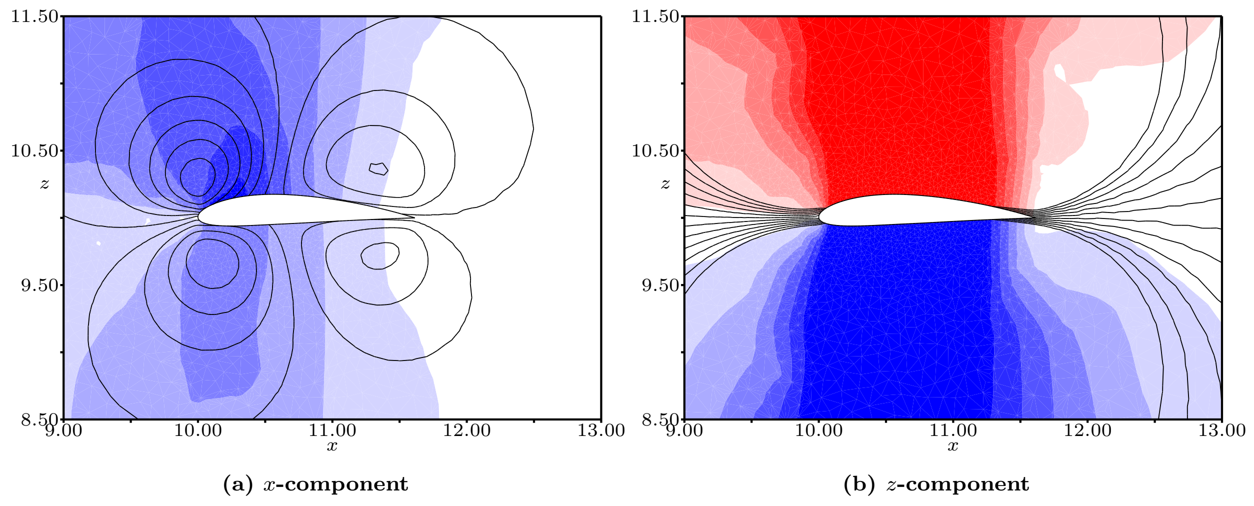


Figure 13. Comparison of sensitivity and displacements for Case #5 on the symmetry plane due to changes in the thickness at the wing root. Color contours are displacements between baseline and perturbed configuration. Line are contours of analytically-computed sensitivity.

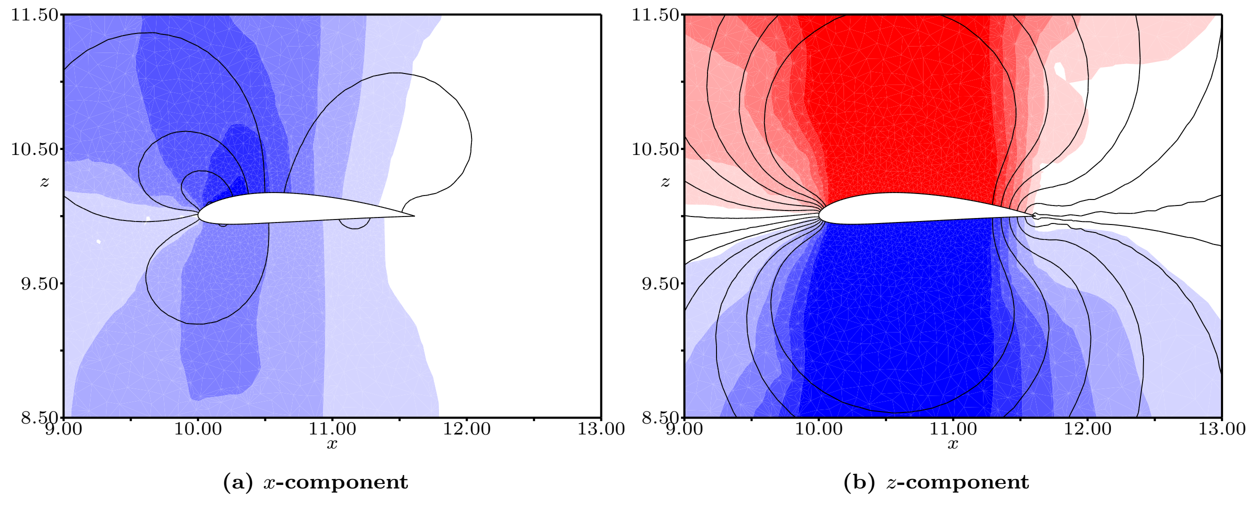
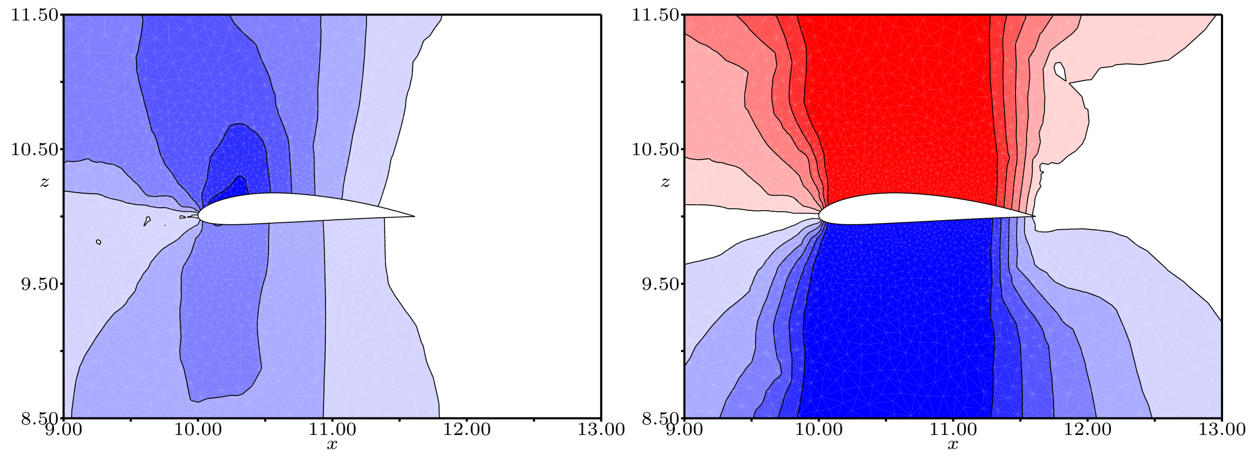


Figure 14. Comparison of sensitivity and displacements for Case #6 on the symmetry plane due to changes in the thickness at the wing root. Color contours are displacements between baseline and perturbed configuration. Line are contours of analytically-computed sensitivity.



(a) *x*-**component** **(b)** *z*-**component**

Figure 15. Comparison of sensitivity and displacements for Case #7 on the symmetry plane due to changes in the thickness at the wing root. Color contours are displacements between baseline and perturbed configuration. Line are contours of analytically-computed sensitivity.

Table 2. Optimized design variables produced by the `FUN3D` design framework.

Design Variable	Original	Case #1	Case #2	Case #3	Case #4	Case #5	Case #6	Case #7
thick[1]	0.14000	0.13968	0.13952	0.13969	0.13968	0.13952	0.13969	0.14004
thick[2]	0.12000	0.11970	0.11970	0.11970	0.11970	0.11970	0.11970	0.12050
thick[3]	0.10000	0.09994	0.09994	0.09994	0.09994	0.09994	0.09994	0.10025
thick[4]	0.08000	0.08001	0.08001	0.08001	0.08001	0.08001	0.08001	0.08007
thick[5]	0.06000	0.06006	0.06006	0.06006	0.06006	0.06006	0.06006	0.06011
thick[6]	0.04000	0.04000	0.04000	0.04000	0.04000	0.04000	0.04000	0.04001
camber[1]	0.04000	0.03450	0.03450	0.03450	0.03450	0.03450	0.03450	0.03454
camber[2]	0.04000	0.02766	0.02765	0.02765	0.02766	0.02765	0.02765	0.02762
camber[3]	0.03000	0.02783	0.02783	0.02783	0.02783	0.02783	0.02783	0.02779
camber[4]	0.02000	0.02633	0.02634	0.02633	0.02633	0.02634	0.02633	0.02631
camber[5]	0.01000	0.02100	0.02100	0.02100	0.02100	0.02101	0.02100	0.02102
camber[6]	0.00000	0.00415	0.00415	0.00415	0.00415	0.00415	0.00415	0.00416
incidence[1]	0.00000	-0.00049	-0.00048	-0.00048	-0.00049	-0.00048	-0.00048	-0.00049
incidence[2]	0.00000	-0.00148	-0.00148	-0.00148	-0.00148	-0.00148	-0.00148	-0.00150
incidence[3]	0.00000	-0.00047	-0.00047	-0.00047	-0.00047	-0.00047	-0.00047	-0.00048
incidence[4]	0.00000	0.00055	0.00055	0.00055	0.00055	0.00055	0.00055	0.00054
incidence[5]	0.00000	0.00078	0.00078	0.00078	0.00078	0.00078	0.00078	0.00078
incidence[6]	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

upon these results, it appears that the discrepancies between the sensitivities and displacements do not adversely impact either the quality of the design results, nor the convergence rate, of the optimizer.

The final state of the design variables for each case can be seen in Table 2.

Table 3. Optimized design variable state produced by the `CART3D` design framework.

Design Variable	Original	`CART3D`
thick[1]	0.14000	0.16421
thick[2]	0.12000	0.16274
thick[3]	0.10000	0.13878
thick[4]	0.08000	0.11257
thick[5]	0.06000	0.09308
thick[6]	0.04000	0.04548
camber[1]	0.04000	0.03232
camber[2]	0.04000	0.02594
camber[3]	0.03000	0.02505
camber[4]	0.02000	0.01965
camber[5]	0.01000	0.01071
camber[6]	0.00000	0.00060
incidence[1]	0.00000	-0.22570
incidence[2]	0.00000	-0.23151
incidence[3]	0.00000	0.56050
incidence[4]	0.00000	0.69111
incidence[5]	0.00000	1.07874
incidence[6]	0.00000	0.41552

B. Treatment of sensitivities in the `CART3D` environment

Unlike `FUN3D`, the `CART3D` design framework deals with the mesh motion internally. Because of this there are no "displacement method" options. Where `FUN3D` needs a consistent mesh topology throughout the design study, `CART3D` does not. In fact, every design iteration produces a new mesh – any new shape will result in the discrete body cutting through the Cartesian mesh in a different manner. The use of a new mesh in every design iteration introduces a slightly different level of discretization error that, in some cases, may contribute to a higher level of numerical noise in the optimization. This potential problem is avoided through the use of adjoint-based mesh refinement that tightly controls the level of discretization error in every design cycle. This `CART3D` option was used in the design study.

VR-MVC was used as the "sensitivity method" to propagate BRep Edge movements into the interior of the BRep Face. But for this design study, MVC alone would produce the same results. This is because the symmetry plane is handled by `CART3D` and there is visibility within the bounds of all of the wing surfaces. Therefore only a single design case which used analytic sensitivities was run and the results can be seen in Table 3.

It must be noted that the objective here is to *demonstrate* the use if `ESP`'s sensitivity calculations in both the `FUN3D` and `CART3D` design environments. These results should NOT be used to compare the two environments, since they use different optimizers and different flow field resolutions. Even so, the two schemes show similar design trends, such as the incidences are almost the same, the thicknesses are increasing, and the cambers are moving in similar directions.

VI. Conclusions

The Engineering Sketch Pad (ESP) has been integrated into both the CART3D and FUN3D design frameworks. When applied to an inviscid wing, both frameworks were able to decrease the (induced) drag coefficient by about 5.5%, while holding the lift coefficient fixed. This was done in both frameworks by allowing the optimizer to change the spanwise thickness, camber, and incidence distributions.

The initial application of ESP revealed that the method used to propagate boundary changes to the interior of a Face, mean value coordinates (MVC), could produce results that were not intuitive; these results were due to the fact that MVC guarantees smoothness but does not guarantee that the maximum sensitivity occurs on the boundary, but may in fact occur in the interior of the region. This undesirable result comes about because MVC can produce negative weights when the interpolation point is not visible to some the portion of the boundary. To circumvent this, a new visibility-restricted (VR-MVC) was developed and applied.

As an alternative to either MVC or VR-MVC, the Delaunay-type surface tessellator was directly differentiated, as described above. The key to computing sensitivities in this way is the fact that only formulae that produce the locations of the interior vertices (u, v) need to be differentiated, and so this technique should be rather straightforward to apply to existing tessellators.

Notwithstanding the above, differences were observed between the contours of the predicted sensitivities and contours of displacements (which were computed via finite differences). The key reason for these differences was the way in which the sensitivities are computed along Edges. In particular, the sensitivity of points on Edges is due to three effects: the shape of the Edge changing, changes in the parametric coordinates at the ends of the Edge (due to trimming effects), and the spacing of the points along the Edge. The first two of these are treated in a completely consistent way, as seen by noting that the shapes of the boundary curves are essentially the same. But the last effect, which establishes the distribution of the points along the Edge, is not consistent because the sensitivity calculation assumes a motion normal to the Edge whereas the displacements assume that the relative arc-length spacing does not change; the relative-arc-length-spacing strategy can only be done via regenerating the configuration.

In order to assess the importance of this discrepancy, several different test cases were computed within the FUN3D framework. In each case, the optimized result was nearly identical (within 0.1%). The case in which the sensitivity and displacements agreed along the Edges, and hence Faces, required about 5% fewer flow solver iterations to achieve its result, but at the expense of finite differences, which require a regeneration for each design variable at each optimization step (as opposed to only one regeneration for the analytical sensitivities). It is useful to reiterate that the discrepancies seen are due to discrete points *sliding* along the owning geometric entities.

Although the detailed investigation was not pursued using the CART3D framework, it too demonstrated that the analytical sensitivities were sufficient to achieve optimized results that were comparable with the FUN3D results.

Acknowledgement

This work was funded by the CAPS project, AFRL Contract FA8050-14-C-2472: “CAPS: Computational Aircraft Prototype Syntheses”; E.J. Alyanak is the Technical Monitor. The description of the “CART3D Design Framework”, (section III) was contributed by Marian Nemac (Science & Technology Corp. at NASA Ames); he played a key role in the coupling of ESP to the CART3D Design Framework. The description of the “FUN3D Design Framework” (section IV) was contributed by W.T. Jones (NASA Langley); he was also instrumental in the coupling of ESP to the FUN3D Design Framework.

References

[1]Nemec, M., and Aftosmis, M.J., "Parallel Adjoint Framework for Aerodynamic Shape Optimization of Component-Based Geometry", AIAA-2011-1249, Jan. 2011.

[2]Nielsen, E.J., "Adjoint-Based Aerodynamic Design of Complex Aerospace Configurations", ASME 2016-7573, July 2016.

[3]Palacios, F., Economon, T.D., Wendroff, A.D., and Alonso, J.J., "Large-scale aircraft design using SU2", AIAA-2015-1946, January 2015.

[4]Samareh, J.A., "Aerodynamic Shape Optimization Based on Free-form Deformation", AIAA-2004-4630, September 2004.

[5]Samareh, J.A., "Multidisciplinary Aerodynamic-structural Shape Optimization Using Deformation (MASSOUD)", AIAA-2000-4911, June 2000.

[6]Haimes, R. and Dannenhoffer, J.F., "The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry", AIAA-2013-3073, June 2013.

[7]Dannenhoffer, J.F., and Haimes, R., "Design Sensitivity Calculations Directly on CAD-based Geometry", AIAA-2015-1370, January 2015.

[8]Hormann, K, and Floater, M.S., "Mean Value Coordinates for Arbitrary Planar Polygons", *ACM Transactions on Graphics*, Vol. 24, No. 4, October 2006, pages 1424–1441.

[9]Lipman, Y., Kopf, J., Cohen-Or, D., and Levin, D., "GPU-assisted Positive Mean Value Coordinates for Mesh Deformations", Eurographics Symposium on Geometry Processing, 2007.

[10]Bowyer, A., "Computing Dirichlet tessellations", *Computer Journal*, V. 24, N. 2, pp 162–166, 1981.

[11]Watson, D.F., "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes", *Computer Journal*, V. 24, N. 2, pp 167–172, 1981.

[12]Gill, P.E., Murray, W, and Saunders, M.A., "SNOPT: An SQP algorithm for large-scale constrained optimization", *SIAM Journal on Optimization*, V. 12, 1997.

[13]Nemec, M., and Aftosmis, M.J.,"Adjoint Sensitivity Computations for an Embedded-Boundary Cartesian Mesh Method", *Journal of Computational Physics*, V. 227, pp 2724–2742, 2008.

[14]Nemec, M., and Aftosmis, M.J., "Adjoint Algorithm for CAD-Based Shape Optimization Using a Cartesian Method", AIAA-2005-4987, June 2005.

[15]Anderson, G.R., Nemec, M., and Aftosmis, M.J., "Aerodynamic Shape Optimization Benchmarks with Error Control and Automatic Parameterization", AIAA-2015-1719, Jan. 2015.

[16]Rodriguez, D.L., Aftosmis, M.J., Nemec, M., and Anderson, G.R., "Optimization of Flexible Wings with Distributed Flaps at Off-Design Conditions", *AIAA Journal*, (2016), accessed Nov. 22, 2016. doi: http://dx.doi.org/10.2514/1.C033535

[17]Nielsen, E. J., and Park, M. A., "Using an Adjoint Approach to Eliminate Mesh Sensitivities in Computational Design", *AIAA Journal*, Vol. 44, No. 5, May 2006, pp. 948–953.

[18]Anderson, W. K., and Bonhaus, D. L., "An Implicit Upwind Algorithm for Computing Turbulent Flows on Unstructured Grids", *Computers and Fluids*, Vol. 23, No. 1, 1994, pp. 1–21, doi: 10.1016/0045-7930(94)90023-X.

[19]Anderson, W. K., Rausch, R. D., and Bonhaus, D. L., "Implicit/Multigrid Algorithms for Incompressible Turbulent Flows on Unstructured Grids", *Journal of Computational Physics*, Vol. 128, No. 2, 1996, pp. 391–408, doi: 10.1006/jcph.1996.0219.

[20]Nielsen, E. J., "Aerodynamic Design Sensitivities on an Unstructured Mesh Using the Navier-Stokes Equations and a Discrete Adjoint Formulation", *Ph.D. thesis*, Virginia Polytechnic Institute and State University, Department of Aerospace and Ocean Engineering, Dec, 1998.

[21]Biedron, R. T., and Thomas, J. L., "Recent Enhancements to the FUN3D Flow Solver for Moving-Mesh Applications", AIAA 2009-1360, Jan, 2009.

[22]Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes", *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372, doi: 10.1016/0021-9991(81)90128-5.

[23]Saad, Y. and Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems", *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3, 1986, pp. 856–869, doi: 10.1137/0907058.

[24]Nielsen, E. J. and Anderson, W. K., "Recent Improvements in Aerodynamic Design Optimization on Unstructured Meshes", *AIAA Journal*, Vol. 40, No. 6, 2002, pp. 1155–1163, doi: 10.2514/2.1765.

[25]Nielsen, E. J., Lu, J., Park, M. A., and Darmofal, D. L., "An Implicit, Exact Dual Adjoint Solution Method for Turbulent Flows on Unstructured Grids", *Computers and Fluids*, Vol. 33, No. 9, 2004, pp. 1131–1155, doi: 10.1016/j.compfluid.2003.09.005.

[26]Nielsen, E. J. and Diskin, B., "Discrete Adjoint-Based Design for Unsteady Turbulent Flows on Dynamic Overset Unstructured Grids", *AIAA Journal*, Vol. 51, No. 6, June 2013, pp. 1355–1373, doi: 10.2514/1.J051859.