AIAA JOURNAL
Vol. 59, No. 4, April 2021

Efficient Mesh Generation and Deformation for Aerodynamic Shape Optimization

Ney R. Secco*
Technological Institute of Aeronautics, 12228-900 São José dos Campos, Brazil
and
Gaetan K. W. Kenway,† Ping He,‡ Charles Mader,‡ and Joaquim R. R. A. Martins§
University of Michigan, Ann Arbor 48109, Michigan

https://doi.org/10.2514/1.J059491

Mesh generation and deformation are critical elements in gradient-based aerodynamic shape optimization (ASO). Improperly generated or deformed meshes may contain bad-quality cells that degrade the accuracy of computational fluid dynamics (CFD) solvers. Moreover, an inefficient mesh deformation method can become the bottleneck for the entire ASO process. To perform practical ASO, mesh generation and deformation methods need to be automated, scalable, robust, and computationally efficient. This paper tackles these challenges by developing an efficient approach for generating high-quality structured meshes in a semi-automatic manner. An automatic mesh generation approach is also proposed to handle intersections of multiple structured meshes with the overset mesh approach. In addition to mesh generation, a flexible mesh deformation method is developed, along with an efficient approach for computing mesh deformation derivatives using automatic differentiation. Finally, the performance of the proposed approaches is evaluated in terms of speed, scalability, and robustness. The mesh generation approach scales up to 100 million cells and 256 CPU cores. In addition, the robust mesh deformation approach enables a large range of valid mesh deformations, which gives more freedom to explore the design space in ASO. Moreover, the mesh deformation and the computation of its derivatives require only 0.1% of the CFD runtime. The mesh generation and deformation approaches have been implemented in the pyHyp and IDWarp software packages, which are publicly available under open-source licenses. The proposed approaches are useful tools to handle general ASO problems for aircraft, turbomachinery, and ground vehicles.

Nomenclature

a, b	=	exponents controlling the length of near-field deformation regions
$\boldsymbol{b}_i$	=	translation vector induced by point i in mesh deformation
$c_{\max}$	=	parameter controlling the number of pseudomarching steps
L_{def}	=	parameter controlling how far surface deformations propagate in volume meshes
$\boldsymbol{M}_i$	=	rotation matrix induced by point i in mesh deformation
$\boldsymbol{n}$	=	local normal vector in physical coordinates
q	=	mesh growth factor
V	=	mesh cell volume, m^3
v	=	blending factor for mesh marching
$\boldsymbol{x}$	=	mesh point coordinate vector, m
α	=	scaling factor controlling the length of near-field deformation regions
Δd	=	mesh marching distance, m
ΔS	=	mesh cell area, m^2
ϵ_i, ϵ_e	=	dissipation coefficients to stabilize hyperbolic mesh marching
ζ	=	parametric coordinate normal to body surfaces
ξ, η	=	parametric coordinates along body surfaces
σ	=	splay factor to increase the overlapped region among overset meshes

Received 6 February 2020; revision received 24 July 2020; accepted for publication 8 October 2020; published online 11 February 2021.

*Assistant Professor, Division of Aeronautical Engineering, São Paulo.
†Research Investigator, Department of Aerospace Engineering.
‡Assistant Research Scientist, Department of Aerospace Engineering.
§Professor, Department of Aerospace Engineering. Fellow AIAA.

I. Introduction

MESH generation and deformation are vital components in modern computational fluid dynamics (CFD) applications. Generating a mesh that matches a given geometric shape is necessary when running a simple parameter study, a design of experiments, or more complicated applications such as shape optimization or fluid–structure interaction simulations. In the specific case of gradient-based aerodynamic shape optimization (ASO), we need an efficient procedure to deform the mesh as the design surface changes. The mesh deformation also needs to be amenable to the efficient computation of derivative information.

Mesh topology is divided into two categories: structured and unstructured. Compared with structured meshes, unstructured meshes have a flexible connectivity pattern and are therefore more suitable for handling complex geometries. Owing to this advantage, the mesh generation for unstructured meshes can be fully automatic, which dramatically reduces the preprocessing time for CFD [1]. However, the above flexibility comes at a price: unstructured CFD solvers are usually less efficient in terms of memory, execution speed, and numerical convergence than structured solvers [2]. Because ASO usually requires hundreds of CFD simulations, the efficiency of structured CFD solvers is advantageous for such applications. However, the generation of structured meshes is more challenging to automate than unstructured meshes. Fortunately, various mesh extrusion methods have been proposed to generate high-quality structured meshes for simple geometry components, such as a wing or a fuselage [3]. To handle complex geometries that contain multiple components, such as a wing–body–tail configuration, we can use the overset mesh approach, in which multiple structured meshes are combined [2,4–6].

Once the baseline mesh is generated, we need to update it efficiently during the shape optimization process. There are two options for updating the mesh: regeneration and deformation. Mesh regeneration has a higher likelihood of yielding usable meshes for widely different geometries. However, this option applies mainly to unstructured CFD solvers with a fully automatic mesh generation capability. In addition, unstructured mesh regeneration may change the number of mesh cells, as well as mesh topology and connectivity. This introduces additional numerical noise and reduces the smoothness

of the solution space [7,8], which eventually affects the convergence of the optimization. In contrast, mesh deformation techniques preserve the mesh connectivity and provide a smooth variation in the mesh, making them more amenable for gradient-based optimization.

There are two main types of mesh deformation methods: physically based and interpolation based. The physically based methods treat the mesh deformation as elastic or diffusion problems and are intuitive to implement [9–11]. However, they require mesh connectivity information and are not easily extensible to some mesh topologies, such as meshes with hanging nodes [8]. Moreover, physically based methods often suffer from poor deformation convergence due to the stiff system matrix caused by mesh cells with high aspect ratio or skewness. Interpolation-based methods compute the deformation of mesh points based on their wall distances. They do not require mesh connectivity information and easily handle arbitrary mesh types [12–14]. For instance, Melville [15] defined grid response functions based on the wetted surface displacement to consistently deform overset meshes, preserving their relative positions and interpolation stencils. In addition, the efficiency of interpolation-based methods does not depend on the mesh quality (e.g., skewness, aspect ratio), so they can handle larger mesh deformations. Both physically based and interpolation-based methods have experienced significant development in the last decade [7,8,11,13,16–20].

Despite the efforts cited above, developing effective mesh generation and deformation approaches remains challenging for gradient-based ASO. More specifically, we need a robust way to generate high-quality structured meshes for all the geometry components and their intersections. Furthermore, we need an efficient algorithm that, in addition to deforming the mesh, computes the deformation derivatives. Bischof et al. [21] applied automatic differentiation (AD) to a mesh generation engine to obtain derivatives of the volume mesh coordinates with respect to the chord of a wing. Truong et al. [22] demonstrated the differentiation of a mesh deformation method based on linear elasticity and its application to airfoil optimization. The mesh generation and deformation approaches need to be automated to perform ASO in a scalable, robust, and computationally efficient way.

To tackle the above challenges, we propose a semi-automatic approach to generate high-quality structured meshes for complex geometries with multiple components and a fully automatic approach for robust mesh deformation. In the proposed mesh generation approach, we manually generate the surface mesh for each geometry component and automatically extrude the volume mesh by solving hyperbolic equations [3]. We then combine all the meshes using the overset mesh approach and develop an automatic mesh generation approach to handle their intersections. The above efforts have been implemented in the pyHyp[¶] and pySurf packages.

For the mesh deformation, we implement an approach based on the inverse distance weighting method originally proposed by Luke et al. [8]. Building on their work, we develop modifications that improve the original algorithm's robustness and computational efficiency, along with efficient computation of the mesh derivatives. The mesh deformation method has been implemented in the IDWarp package.[**] Finally, we demonstrate the performance of our implementations in terms of speed, scalability, and robustness, and illustrate their applications for ASO with structured and unstructured CFD solvers (ADflow [23] and DAFoam [24]). The pyHyp and IDWarp packages are available under open-source licenses and have been extensively used in our high-fidelity ASO framework, MACH-Aero.[††]

The rest of this paper is organized as follows. In Secs. II and III, we detail the theoretical background for mesh generation and deformation, respectively. The performance of the proposed mesh generation and deformation approaches is demonstrated and discussed in Sec. IV. We summarize our findings in Sec. V.

[¶]https://github.com/mdolab/pyhyp.

[**]https://github.com/mdolab/idwarp.

[††]https://github.com/mdolab/mach-aero.

II. Mesh Generation Methods

In this section, we first elaborate on our semi-automatic mesh generation approach for a single geometry component. As mentioned in Sec. I, to enable structured meshes for multiple components, we use the overset mesh approach, reported in our previous work [2]. To automate the overset mesh generation process, we develop a robust mesh generation approach for intersection [25]. The details of this process are discussed in the second part of this section.

A. Semi-Automatic Mesh Generation for a Single Geometry Component

Our semi-automatic mesh generation approach starts with a manual generation of structured surface meshes. For instance, we can use the mesh generation tool ICEM-CFD [26] to import a baseline computer-aided design (CAD) geometry and build structured surface meshes over it. The surface mesh topology can be independent of the underlying CAD surface patches because the ICEM-CFD mesh generation procedure only uses the CAD patches to project the desired mesh topology over the reference surface. Then, we extrude the surface meshes into volume meshes. The volume mesh extrusion is automated by using the hyperbolic mesh marching method proposed by Chan and Steger [3]. This process generates the volume mesh layer-by-layer along the direction normal to the surface. We illustrate the mesh extrusion for a simple wing geometry in Fig. 1.

Let $\boldsymbol{x} = \boldsymbol{x}(\xi, \eta, \zeta)$ be a mapping from the parametric space coordinates ξ, η, and ζ to the physical space coordinate vector $\boldsymbol{x} = [x, y, z]$. The ξ and η parametric coordinates lie along the body surface and ζ is the mesh marching direction (normal to the surface), as shown in Fig. 2. The physical coordinates of the mesh points are the image of a regular grid constructed in the parametric space given by this mapping, which is built upon the following set of differential equations [27]:

$$\boldsymbol{x}_\xi \cdot \boldsymbol{x}_\zeta = 0 \tag{1a}$$

$$\boldsymbol{x}_\eta \cdot \boldsymbol{x}_\zeta = 0 \tag{1b}$$

$$\boldsymbol{x}_\zeta \cdot (\boldsymbol{x}_\xi \times \boldsymbol{x}_\eta) = \Delta V \tag{1c}$$

where the vectors $\boldsymbol{x}_\xi$, $\boldsymbol{x}_\eta$, and $\boldsymbol{x}_\zeta$ are the partial derivatives of $\boldsymbol{x}$ with respect to ξ, η, and ζ, respectively. The Jacobian cell volume (ΔV) is a field parameter that controls the size of cell volumes throughout the mesh. Equations (1a) and (1b) impose orthogonal constraints to the

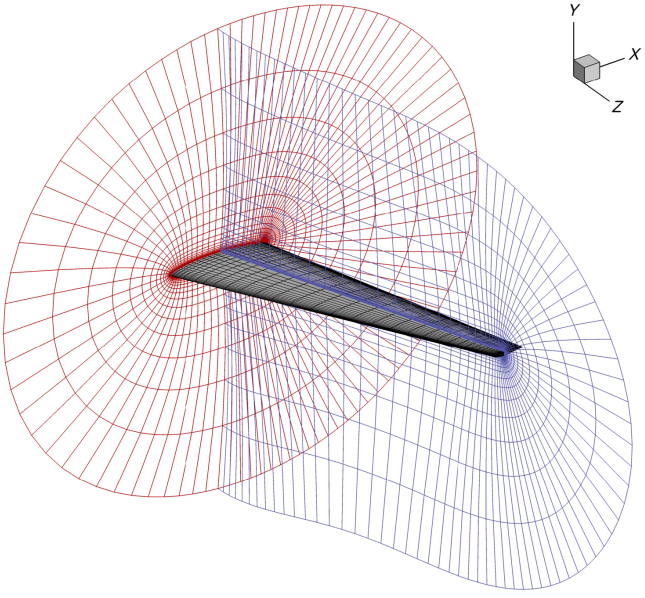


Fig. 1 Example of volume mesh extrusion. The surface mesh is shown in gray and two slices of volume mesh are shown in red and blue.

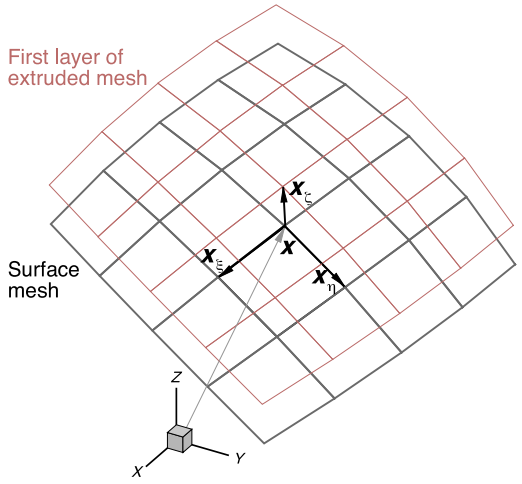


Fig. 2 System of coordinates used in the hyperbolic mesh extrusion.

extruded mesh point, and Eq. (1c) imposes a volume constraint to control the distance of mesh extrusion.

Following Chan and Steger [3], we linearize Eqs. (1a–1c) around a nearby mesh point, which leads to a linear system of size $3n_p$, where n_p is the number of points on the surface patch:

$$\boldsymbol{P}\cdot\Delta\boldsymbol{x}_{i+1,j,k}+\boldsymbol{Q}\cdot\Delta\boldsymbol{x}_{i-1,j,k}+\boldsymbol{R}\cdot\Delta\boldsymbol{x}_{i,j+1,k}+\boldsymbol{S}\cdot\Delta\boldsymbol{x}_{i,j-1,k}+\boldsymbol{T}\cdot\Delta\boldsymbol{x}_{i,j,k}=\boldsymbol{g} \tag{2}$$

The indices i, j, and k are the mesh parametric indices for ξ, η, and ζ, respectively. Solving the above equation yields the displacements $\Delta\boldsymbol{x}$ that determine the next mesh layer. The matrices in this equation are defined as

$$\begin{array}{l|l|l}\boldsymbol{P}=\dfrac{1+\theta}{2}\boldsymbol{C}^{-1}\boldsymbol{A}-\epsilon_{i\xi}\boldsymbol{I} & \boldsymbol{Q}=-\dfrac{1+\theta}{2}\boldsymbol{C}^{-1}\boldsymbol{A}-\epsilon_{i\xi}\boldsymbol{I} & \boldsymbol{R}=\dfrac{1+\theta}{2}\boldsymbol{C}^{-1}\boldsymbol{B}-\epsilon_{i\eta}\boldsymbol{I}\\ \hline \boldsymbol{S}=-\dfrac{1+\theta}{2}\boldsymbol{C}^{-1}\boldsymbol{B}-\epsilon_{i\eta}\boldsymbol{I} & \boldsymbol{T}=(1-2\epsilon_{i\xi}-2\epsilon_{i\eta})\boldsymbol{I} & \boldsymbol{g}=\boldsymbol{C}^{-1}\begin{bmatrix}0\\0\\\Delta V_{i,j,k}\end{bmatrix}+\boldsymbol{D}_e\end{array} \tag{3}$$

where $\boldsymbol{I}$ is a 3×3 identity matrix and

$$\boldsymbol{A}=\begin{bmatrix}\boldsymbol{x}_\zeta\\\boldsymbol{0}\\\boldsymbol{x}_\eta\times\boldsymbol{x}_\zeta\end{bmatrix}\quad\boldsymbol{B}=\begin{bmatrix}\boldsymbol{0}\\\boldsymbol{x}_\zeta\\\boldsymbol{x}_\zeta\times\boldsymbol{x}_\xi\end{bmatrix}\quad\boldsymbol{C}=\begin{bmatrix}\boldsymbol{x}_\xi\\\boldsymbol{x}_\eta\\\boldsymbol{x}_\xi\times\boldsymbol{x}_\eta\end{bmatrix} \tag{4}$$

Chan and Steger [3] provided a complete description of the dissipation factors θ, $\epsilon_{i\xi}$, $\epsilon_{i\eta}$, and $\boldsymbol{D}_e$. The hyperbolic marching equations per se enforce the orthogonality of the volume cells with respect to the surface, but this may cause the generation of negative volume cells, especially when extruding concave surfaces. Therefore, dissipation terms are necessary to relax the orthogonality requirement and spread the marching directions. We use small dissipation factors for the first layers to keep boundary-layer cells orthogonal to the wall. Then, we steadily increase these factors to prevent the folding of the marching front. Even though the discretized partial differential equation (PDE) of Eq. (2) is not strictly hyperbolic due to the dissipation terms, the *hyperbolic mesh generation* term comes from the fact that there is no need to enforce boundary conditions at the outer layer of the mesh [28].

The cell volume parameter $\Delta V_{i,j,k}$ is given by

$$\Delta V_{i,j,k}=S_{i,j,k}\cdot\Delta d_k \tag{5}$$

where $S_{i,j,k}$ is an area value associated with node (i,j); $S_{i,j,k}$ is calculated by averaging the areas of the quadrilaterals that share this node at the current layer k, and Δd_k is the average marching distance (cell height) desired for the current layer.

During the initialization of the mesh marching process, user specify the number of layers n_k, the marching distance of the first layer Δd_1, and the desired marching distance d for the entire surface mesh. Assuming a geometric progression of cell sizes, we can relate the average cell heights of two consecutive layers as

$$\Delta d_k=q\cdot\Delta d_{k-1} \tag{6}$$

where $q>1$ is a constant growth factor. The sum of all average cell distances between layers should be equal to the overall marching distance of the mesh:

$$d=\sum_{k=1}^{n_k-1}\Delta d_k=\Delta d_1\frac{1-q^{n_k-1}}{1-q} \tag{7}$$

This can be rearranged to obtain the following equation:

$$\Delta d_1(1-q^{n_k-1})-d(1-q)=0 \tag{8}$$

which we solve for q using Newton's method. Once we have q, we can determine the average cell height Δd_k of every layer with Eq. (6).

The derivatives $\boldsymbol{x}_\xi$ and $\boldsymbol{x}_\eta$ are computed using central differences (assuming $\Delta\xi=\Delta\eta=1$):

$$\boldsymbol{x}_\xi=\frac{\boldsymbol{x}_{i+1,j}-\boldsymbol{x}_{i-1,j}}{2}\qquad\boldsymbol{x}_\eta=\frac{\boldsymbol{x}_{i,j+1}-\boldsymbol{x}_{i,j-1}}{2} \tag{9}$$

and $\boldsymbol{x}_\zeta$ is computed by solving Eqs. (1a–1c) at the linearization point:

$$\boldsymbol{x}_\zeta=\begin{bmatrix}\boldsymbol{x}_\xi\\\boldsymbol{x}_\eta\\\boldsymbol{x}_\xi\times\boldsymbol{x}_\eta\end{bmatrix}^{-1}\cdot\begin{bmatrix}0\\0\\\Delta V_{i,j,k-1}\end{bmatrix}=\boldsymbol{C}^{-1}\cdot\begin{bmatrix}0\\0\\\Delta V_{i,j,k-1}\end{bmatrix} \tag{10}$$

Once we implicitly solve Eq. (2) for all points of layer k, we get the coordinates of the next layer with $\boldsymbol{x}_{i,j,k+1}=\boldsymbol{x}_{i,j,k}+\Delta\boldsymbol{x}_{i,j}$. We use the generalized minimal residual (GMRES) solver from the PETSc [29] library to solve this linear system.

Regular points in a structured surface mesh have four neighbors. However, if a structured surface mesh is composed of multiple patches, irregular points with a distinct number of neighbors may appear at corners shared by multiple patches, such as in O-meshes (Fig. 3). The central difference formulations in Eq. (9) no longer apply for these irregular points. To tackle this issue, we approximate $\boldsymbol{x}_\xi$ and $\boldsymbol{x}_\eta$ for an irregular point located at $\boldsymbol{x}_0$ with M neighbors as

$$\begin{aligned}\boldsymbol{x}_\xi&=\frac{2}{M}\sum_{i=1}^{M}(\boldsymbol{x}_i-\boldsymbol{x}_0)\cos\left(\frac{2\pi(i-1)}{M}\right)\\ \boldsymbol{x}_\eta&=\frac{2}{M}\sum_{i=1}^{M}(\boldsymbol{x}_i-\boldsymbol{x}_0)\sin\left(\frac{2\pi(i-1)}{M}\right)\end{aligned} \tag{11}$$

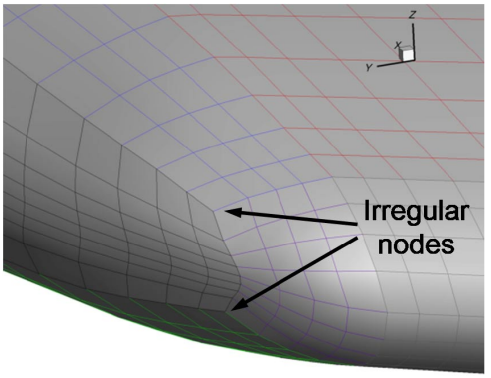


Fig. 3 Irregular points in a structured O-mesh topology around a wingtip.

where $\boldsymbol{x}_i$ represents the location of the ith neighbor. Equation (11) is derived by applying the Taylor series to a regular polygon of radius one centered at $\boldsymbol{x}_0$. Second derivatives can also be computed using

$$\boldsymbol{x}_{\xi\xi} = \frac{2}{M}\sum_{i=1}^{M}(\boldsymbol{x}_i - \boldsymbol{x}_0)\left(4\cos^2\left(\frac{2\pi(i-1)}{M}\right) - 1\right)$$
$$\boldsymbol{x}_{\eta\eta} = \frac{2}{M}\sum_{i=1}^{M}(\boldsymbol{x}_i - \boldsymbol{x}_0)\left(4\sin^2\left(\frac{2\pi(i-1)}{M}\right) - 1\right) \tag{12}$$

It is also necessary to enforce boundary conditions at the edges of the surface patches to compute the derivatives $\boldsymbol{x}_\xi$ and $\boldsymbol{x}_\eta$ at the edge points. We achieve this by creating a layer of ghost points around the surface patches, as shown in Fig. 4.

The position of these ghost points is determined such that the derivatives at edge points (computed with central differences) yield the desired boundary condition effect. For instance, as shown in Fig. 4, a mesh symmetry plane condition around the y plane can be enforced by setting the ghost point at

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \boldsymbol{x}_1 + \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \boldsymbol{x}_2 \tag{13}$$

Since we only solve Eq. (2) for the interior and edge points, it is also necessary to modify this relationship to implicitly include displacements of the ghost points. For example, suppose that (i, j) is an edge point and $(i-1, j)$ is a ghost point. The symmetry plane boundary condition around the y plane containing point (i, j) requires that

$$\Delta\boldsymbol{x}_{i-1,j} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \Delta\boldsymbol{x}_{i+1,j} \tag{14}$$

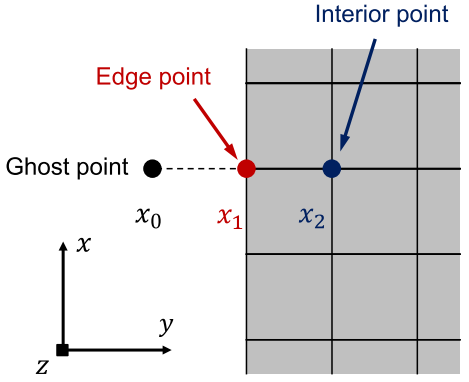


Fig. 4 Ghost points added to the edges of surface patches.

Substituting Eq. (14) into Eq. (2) yields the contribution of point (i, j) to the complete linear system of the problem:

$$\left(\boldsymbol{P} + \boldsymbol{Q} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) \cdot \Delta\boldsymbol{x}_{i+1,j} + \boldsymbol{R} \cdot \Delta\boldsymbol{x}_{i,j+1} + \boldsymbol{S} \cdot \Delta\boldsymbol{x}_{i,j-1} + \boldsymbol{T} \cdot \Delta\boldsymbol{x}_{i,j} = \boldsymbol{g} \tag{15}$$

When dealing with overset meshes, some surface patches have free edges, in the sense that they are not directly connected to symmetry planes or other surface patches. We can force these free edges to splay outward by setting the ghost point as follows:

$$\boldsymbol{x}_0 = \boldsymbol{x}_1 + (1+\sigma)(\boldsymbol{x}_1 - \boldsymbol{x}_2) \tag{16}$$

where σ is the splay factor. The outward splay is useful for increasing the overlapped region among overset meshes, facilitating the hole cutting process. Chan and Steger [3] proposed that the ghost points of a splay boundary condition should satisfy

$$\Delta\boldsymbol{x}_0 = \Delta\boldsymbol{x}_1 + \sigma(\Delta\boldsymbol{x}_1 - \Delta\boldsymbol{x}_2) \tag{17}$$

Assuming the same configurations of points used to derive Eq. (15), the modified version of Eq. (2) that accounts for this relationship is

$$(\boldsymbol{P} - \sigma\boldsymbol{Q}) \cdot \Delta\boldsymbol{x}_{i+1,j} + \boldsymbol{R} \cdot \Delta\boldsymbol{x}_{i,j+1} + \boldsymbol{S} \cdot \Delta\boldsymbol{x}_{i,j-1} + (\boldsymbol{T} + (1+\sigma)\boldsymbol{Q}) \cdot \Delta\boldsymbol{x}_{i,j} = \boldsymbol{g} \tag{18}$$

Once we position the ghost points on all four sides of the structured patches, we used second-order differences to estimate the derivatives. The ghost points are eliminated from the final mesh that is provided by the mesh generator.

The hyperbolic mesh generation algorithm is easy to automate because users only specify the height of the initial layer of cells, the total marching distance, the number of layers, and the amount of dissipation.

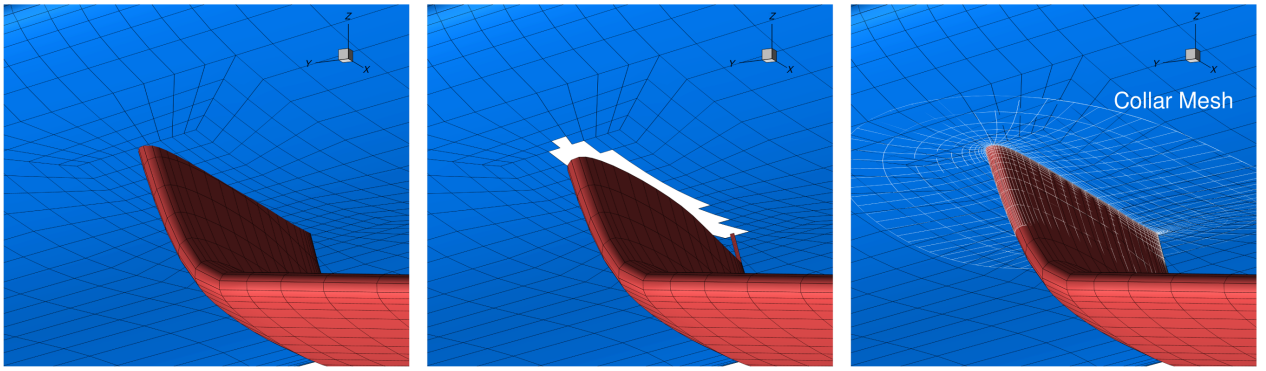
As mentioned in Sec. I, the above mesh generation approach has been implemented in the open-source module pyHyp.

B. Automatic Collar Mesh Generation for Intersecting Geometry Components

As mentioned above, we combine multiple structured meshes using the overset approach to handle complex geometries. However, intersections among multiple geometry components introduce challenges to the generation of high-quality overset meshes. In these cases, *collar meshes* are necessary to ensure that we have cell edges outlining the intersection curves [30], since cells from a primary mesh might be partially covered by cells from other primary meshes (Fig. 5a). These partially covered cells are unsuitable for the finite-volume CFD computations because they are not entirely within the flow domain. These cells should be removed by the overset hole cutting process [2], but their removal would leave a gap in the surface definition (Fig. 5b). Therefore, we add a third overset mesh (the collar mesh) to cover the gap with valid cells (Fig. 5c). In addition to meshing consideration, the collar mesh helps better resolve the boundary-layer flow at the intersection.

The surface mesh generation procedure for overset meshes is subdivided into two parts. First, we manually generate structured surface meshes for each primary component, such as wing, fuselage, and tail. Second, we generate collar meshes around the intersections of the primary meshes.

Collar meshes can be manually generated, just like the primary meshes. However, optimization problems may require the capability to change the intersection geometry, such as moving the wing with respect to the fuselage. In these cases, we use an automated collar mesh generation technique based on hyperbolic equations we previously developed [25]. This technique takes a given curve along



a) Intersecting primary meshes b) Removal of invalid partially covered cells c) Collar mesh outlines the intersection curve

Fig. 5 Collar meshes are necessary to obtain valid cells near intersections.

a reference surface and generates the surface mesh in a layer-by-layer fashion, marching from the curve. Then, we can use pyHyp to extrude the volume mesh based on the collar surface mesh. The hyperbolic equations for surface mesh generation are as follows [31]:

$$\boldsymbol{x}_\xi \cdot \boldsymbol{x}_\eta = 0 \tag{19a}$$

$$\boldsymbol{n} \cdot (\boldsymbol{x}_\xi \times \boldsymbol{x}_\eta) = \Delta S \tag{19b}$$

$$\boldsymbol{n} \cdot \boldsymbol{x}_\eta = 0 \tag{19c}$$

where ξ is the parametric coordinate of the mesh along the reference curve and η is the parametric coordinate along the marching direction, as shown in Fig. 6. The vector $\boldsymbol{n} = (n_1, n_2, n_3)^T$ is the local normal vector (in physical coordinates) from the reference surface onto which the mesh is generated. The scalar factor ΔS controls the area of the cells in the physical space; its value may vary within the bounds of the parametric space to control mesh resolution. Equation (19a) enforces cell orthogonality, Eq. (19b) controls the cell density, and Eq. (19c) indicates that the marching direction should be orthogonal to the surface normal vector.

We follow Chan and Buning [31] to solve the hyperbolic Eqs. (19) with modifications to improve the robustness of collar mesh generation (e.g., blended guide curves marching). We first linearize Eq. (19) around a given reference layer (constant ξ) of the mesh, resulting in a linear system of size $3n_c$, where n_c is the number of points along the reference curve. The contribution of point i of the jth layer to the full linear system is given by

$$\boldsymbol{L}_{i,j} \cdot \Delta\boldsymbol{x}_{i-1,j} + \boldsymbol{M}_{i,j} \cdot \Delta\boldsymbol{x}_{i,j} + \boldsymbol{N}_{i,j} \cdot \Delta\boldsymbol{x}_{i+1,j} = \boldsymbol{f}_{i,j} \tag{20}$$

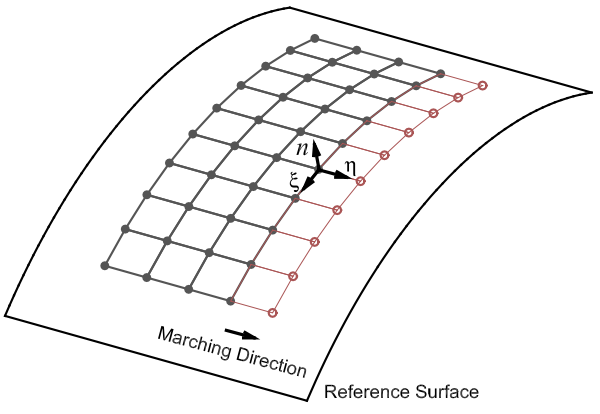


Fig. 6 System of coordinates used in the surface mesh extrusion.

where

$$\boldsymbol{L}_{i,j} = -\frac{\boldsymbol{B}_{i,j}^{-1} \cdot \boldsymbol{A}_{i,j}}{2} - \epsilon_i \boldsymbol{I} \qquad \boldsymbol{M}_{i,j} = (1 + 2\epsilon_i)\boldsymbol{I}$$
$$\boldsymbol{N}_{i,j} = \frac{\boldsymbol{B}_{i,j}^{-1} \cdot \boldsymbol{A}_{i,j}}{2} - \epsilon_i \boldsymbol{I} \qquad \boldsymbol{f}_{i,j} = \boldsymbol{B}_{i,j}^{-1} \cdot \boldsymbol{g}_{i,j} + \epsilon_e \boldsymbol{x}_{\xi\xi}^0 \tag{21}$$

with

$$\boldsymbol{A}_{i,j} = \begin{bmatrix} \boldsymbol{x}_\eta \\ \boldsymbol{n} \times \boldsymbol{x}_\eta \\ \boldsymbol{0} \end{bmatrix} \quad \boldsymbol{B}_{i,j} = \begin{bmatrix} \boldsymbol{x}_\xi \\ \boldsymbol{n} \times \boldsymbol{x}_\xi \\ \boldsymbol{n} \end{bmatrix} \quad \boldsymbol{g}_{i,j} = \begin{bmatrix} 0 \\ \Delta S_{i,j} \\ 0 \end{bmatrix} \tag{22}$$

The partial derivatives $\boldsymbol{x}_\xi$ and $\boldsymbol{x}_{\xi\xi}$ along the curve can be obtained with central differences by setting $\Delta\xi = 1$:

$$\boldsymbol{x}_\xi = \frac{\boldsymbol{x}_{i+1,j} - \boldsymbol{x}_{i-1,j}}{2} \qquad \boldsymbol{x}_{\xi\xi} = \boldsymbol{x}_{i+1,j} - 2\boldsymbol{x}_{i,j} + \boldsymbol{x}_{i-1,j} \tag{23}$$

The derivative $\boldsymbol{x}_\eta$ can be obtained by solving Eq. (19) at the linearization point, which yields

$$\boldsymbol{x}_\eta = \begin{bmatrix} \boldsymbol{x}_\xi \\ \boldsymbol{n} \times \boldsymbol{x}_\xi \\ \boldsymbol{n} \end{bmatrix}^{-1} \cdot \begin{bmatrix} 0 \\ \Delta S_{i,j-1} \\ 0 \end{bmatrix} = \boldsymbol{B}_{i,j}^{-1} \cdot \begin{bmatrix} 0 \\ \Delta S_{i,j-1} \\ 0 \end{bmatrix} \tag{24}$$

The dissipation coefficients ϵ_i and ϵ_e are important to stabilize the hyperbolic marching scheme, as discussed in Sec. II.A. Large values of dissipation add stability to the mesh generation process but sacrifice cell orthogonality. Ideally, one should use the smallest dissipation value that ensures a stable solution. To avoid excessive dissipation in regions where it is not necessary, we implement a method to automatically vary the dissipation coefficients. Our method is based on Chan and Steger [3], with modifications that make it suitable for surface mesh generation.

The desired cell area ΔS is given by

$$\Delta S_{i,j} = w_{i,j} \Delta d_j \tag{25}$$

where the average cell width $w_{i,j}$ is the average of the distances of a given point to its two neighbors:

$$w_{i,j} = \frac{|\boldsymbol{x}_{i+1,j} - \boldsymbol{x}_{i,j}| + |\boldsymbol{x}_{i-1,j} - \boldsymbol{x}_{i,j}|}{2} \tag{26}$$

The marching distance Δd_j is computed using the same procedure described for the volume mesh generation, as shown in Eq. (6).

If we apply Eq. (20) to all n_i points of the current layer j, we get a linear system that implicitly yields $\Delta\boldsymbol{x}_{i,j}$ values to determine the points of the next mesh layer:

$$\begin{bmatrix} \boldsymbol{M}_{1,j} & \boldsymbol{N}_{1,j} & \boldsymbol{0} & \cdots & \boldsymbol{L}_{1,j} \\ \boldsymbol{L}_{2,j} & \boldsymbol{M}_{2,j} & \boldsymbol{N}_{2,j} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{L}_{3,j} & \boldsymbol{M}_{3,j} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{N}_{n_i,j} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{M}_{n_i,j} \end{bmatrix} \cdot \begin{bmatrix} \Delta\boldsymbol{x}_{1,j} \\ \Delta\boldsymbol{x}_{2,j} \\ \Delta\boldsymbol{x}_{3,j} \\ \vdots \\ \Delta\boldsymbol{x}_{n_i,j} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_{1,j} \\ \boldsymbol{f}_{2,j} \\ \boldsymbol{f}_{3,j} \\ \vdots \\ \boldsymbol{f}_{n_i,j} \end{bmatrix} \tag{27}$$

We assume that the curve points form a closed loop (points 1 and n_i are neighbors), leading to the off-diagonal elements in the first and last rows of Eq. (27). Once we solve this linear system, we compute the position of the jth layer points using

$$\boldsymbol{x}^*_{i,j+1} = \boldsymbol{x}_{i,j} + \Delta\boldsymbol{x}_{i,j} \tag{28}$$

The new points do not necessarily lie on the reference surface. Therefore, we still need to project these points to the surface to determine the correct position of the next layer of points. This procedure is represented by the projection operator $\mathrm{F}_{\mathrm{proj}}$:

$$\boldsymbol{x}_{i,j+1} = \mathrm{F}_{\mathrm{proj}}(\boldsymbol{x}^*_{i,j+1}) \tag{29}$$

Now we can use $\boldsymbol{x}_{i,j+1}$ as the starting curve for the computation of the next layer by repeating the process.

A surface description is required to compute normal vectors and point projections during the mesh generation. We use triangulated surfaces to represent each primary component, since these surfaces can be easily generated from other surface descriptions, such as B-splines. In addition, we can arbitrarily refine the triangulation near high-curvature regions, such as wing leading edges. In these cases, the triangulated surface resolution should be finer than the surface mesh that will be generated over it, as recommended in previous work that employed the same methodology for overset mesh generation [30,32,33]. Secco et al. [25] discussed the triangulated surface resolution effects on the CFD results of a wing–body geometry.

The use of triangulated surfaces also enables the computation of intersection curves among the primary components. User-specified parameters control the distribution of mesh nodes along the intersection curve to define the first layer for the mesh marching method. The collar mesh is obtained by marching surface meshes on both sides of the intersection curve (Fig. 5c). However, the artificial dissipation associated with the marching scheme tends to smooth node intervals as the march progresses. pySurf has an option to redistribute the nodes of every new layer using the same relative arc-lengths of the first layer of nodes. This is useful to preserve the refinement in specific regions of the collar mesh. Figure 7 shows the application of this procedure to maintain the mesh refinement at the leading and trailing edges of a wing collar mesh.

Users can specify guide curves to control the marching direction of certain points and ensure that relevant surface features, such as sharp corners, are represented by the mesh (see Fig. 7). These guide curves are represented by a set of points connected by the linear segments. At the beginning of the marching process, we detect which points from the baseline curve are the closest ones to each guide curve. Once these points are assigned to their corresponding guide curves, their marching direction is dictated by the curve tangent vector rather than the reference surface normal. Let $\boldsymbol{t}_{g,j}$ be the curve tangent vector at the projection of point $\boldsymbol{x}_{g,j}$ on this curve. Instead of applying the linearization [Eq. (20)] to this point, the displacement vector $\Delta\boldsymbol{x}_{g,j}$ is computed as

$$\boldsymbol{I} \cdot \Delta\boldsymbol{x}_{g,j} = \Delta d_j \cdot \boldsymbol{t}_{g,j} \tag{30}$$

In other words, after solving Eq. (27), we project the point $\boldsymbol{x}_{g,j}$ back to the guide curve rather than the reference surface before marching the next layer. If a guide curve is oblique to the marching direction of the mesh, the hyperbolic equations generate highly skewed or even invalid cells due to the conflicting marching directions. This happens because only the guided point uses the guide curve tangent to compute its displacement (30). We have developed a blending procedure that extends the influence to the guide curve tangent to other points, preventing crossed grid lines.

Let the point $\boldsymbol{x}_{g,j}$ be the point guided by a curve and point $\boldsymbol{x}_{i,j}$ be the point where we apply the blending procedure. We define a user-provided blending factor, ν, that combines the displacement of the guided point and the displacement originally predicted by the hyperbolic marching equations $\Delta\boldsymbol{x}_{i,j}$:

$$\Delta\boldsymbol{x}^b_{i,j} = \nu\Delta\boldsymbol{x}_{g,j} + (1-\nu)\Delta\boldsymbol{x}_{i,j} \tag{31}$$

where $\Delta\boldsymbol{x}^b_{i,j}$ is the blended displacement. If $\nu = 1$, point $\boldsymbol{x}_{i,j}$ marches in a direction parallel to the guide point, whereas no blending occurs for $\nu = 0$. If we substitute $\Delta\boldsymbol{x}_{i,j}$ from Eq. (31) into Eq. (20) we get the blended marching equation

$$\boldsymbol{L}_{i,j} \cdot \Delta\boldsymbol{x}_{i-1,j} + \boldsymbol{M}^b_{i,j} \cdot \Delta\boldsymbol{x}^b_{i,j} + \boldsymbol{N}_{i,j} \cdot \Delta\boldsymbol{x}_{i+1,j} = \boldsymbol{f}^b_{i,j} \tag{32}$$

where we only need to redefine

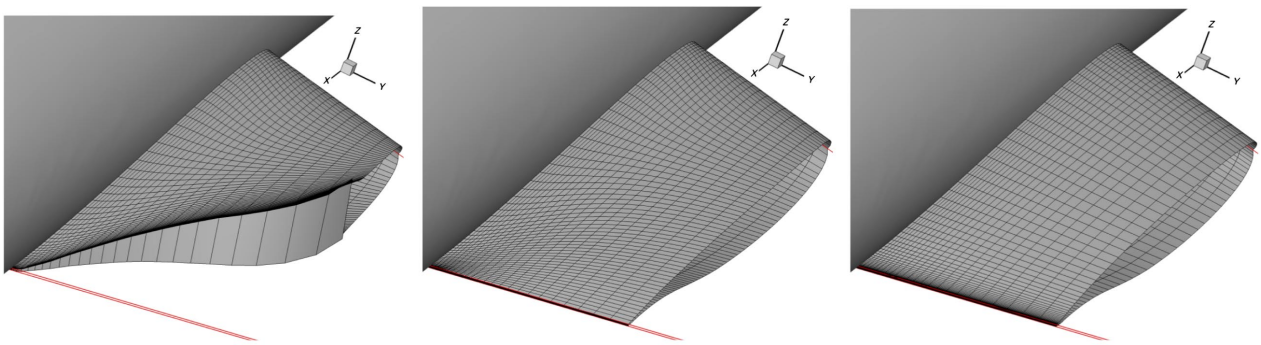
$$\boldsymbol{M}^b_{i,j} = \frac{1}{1-\nu}\boldsymbol{M}_{i,j} \tag{33}$$

$$\boldsymbol{f}^b_{i,j} = \boldsymbol{f}_{i,j} + \frac{\nu}{1-\nu}\boldsymbol{M}_{i,j} \cdot (\Delta d_j \cdot \boldsymbol{t}_{g,j}) \tag{34}$$

We apply the blending to the four nearest neighbors of the guided point. The blending factor is halved for the last two neighbors. Figure 8 shows a blending application for a blunt trailing edge mesh that is marched from an oblique intersection.

Large marching distances may lead to instabilities in the mesh generation process. We mitigate this problem by using pseudomarching steps, where instead of marching directly to the desired marching distance, we execute many intermediate steps until this distance is reached.

The number of intermediate steps to march from layer j to layer $j+1$ is based on the aspect ratio of the cells ($c_{i,j}$), which we define as



a) Original marching scheme **b) Marching with guide curves** **c) Nodal redistribution**

Fig. 7 Additional features implemented in the collar marching scheme to improve mesh quality for CFD analyses.

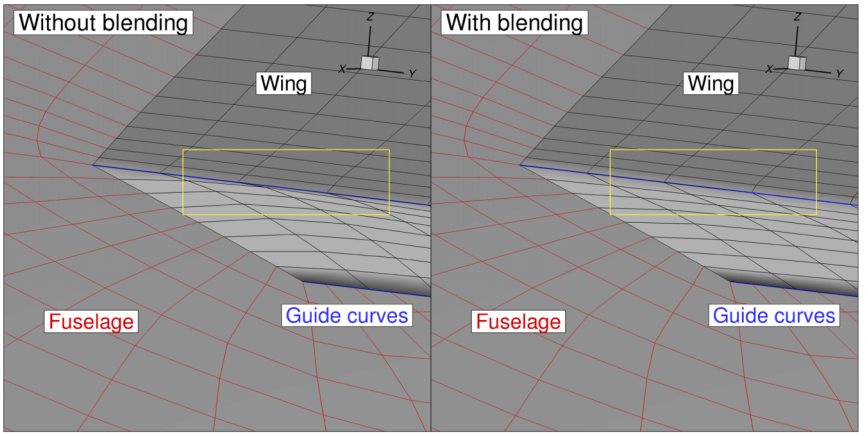


Fig. 8 The blending factor ($\nu = 0.5$) avoids highly skewed cells where guide curves are oblique to the intersection line.

the ratio of their expected height Δd_j, given by Eq. (6), to their width $w_{i,j}$, defined by Eq. (26):

$$c_{i,j} = \frac{\Delta d_j}{w_{i,j}} \tag{35}$$

The number of pseudomarching steps n^* is given by

$$n^* = \left\lceil \frac{\max_i c_{i,j}}{c_{\max}} \right\rceil \tag{36}$$

where $c_{\max}$ is a user-provided value (usually between 5 and 10). Then we compute the pseudomarching distance Δd_p^* as

$$\Delta d_j^* = \Delta d_j / n^* \tag{37}$$

Thus, instead of directly marching to a distance Δd_j, we execute n^* marching steps of size Δd_j^*. The mesh points in these intermediate layers are not included in the final surface mesh. The same pseudomarching scheme is used for the hyperbolic mesh extrusion process of Sec. II.A to increase robustness.

This surface generation process is differentiated to enable gradient-based optimization [25]. The majority of ASO and aerostructural optimization problems have more design variables than functions of interest (objective and constraint functions) [34–36]. In this scenario, reverse AD stands out as an attractive method of derivative computation because its computational cost scales with the number of functions of interest instead of the number of design variables [37]. We use the Tapenade AD engine [38,39] to generate differentiated versions of the code with the source transformation technique. We store some information from the mesh generation process in memory, such as the position of the points before the surface projection and the intermediate layers of the pseudomarching scheme. This avoids recomputing these values when applying the reverse-mode AD, and therefore increases the speed.

Once all the surface meshes are generated, we extrude them into volume meshes using the method described in Sec. II.A. For overset meshes with multiple geometry components, each surface mesh patch is extruded with different marching parameters for better control on the mesh resolution. For instance, we set the initial cell height for the collar meshes to 95% of the cell height of the primary components so that collar cells are preserved during the overset hole cutting process. The total extrusion distance should result in sufficient volume overlap of the overset meshes to allow valid interpolation stencils among them. The recommended extrusion distance ranges from 1 to 3 mean aerodynamic chords.

The current pySurf implementation does not handle cases where three or more geometry components intersect at the same location. One possible solution for handling such cases is to add spider meshes at the corners defined by the corresponding intersections [40].

III. Mesh Deformation Method

In this section, we elaborate on the mesh deformation approach and its integration with a gradient-based optimization framework. The mesh deformation algorithm extrapolates the deformation of surface mesh points to volume mesh points. The extrapolation is based on the inverse distance weighting (IDW) method proposed by Luke et al. [8] with modifications that improve the robustness and computational efficiency, along with efficient computation of mesh deformation derivatives.

We represent the deformation of each surface mesh point i as a combination of a rotation and translation:

$$\boldsymbol{x}_{s,i} = \boldsymbol{M}_i \boldsymbol{x}_{s,i}^{(0)} + \boldsymbol{b}_i \tag{38}$$

where $\boldsymbol{x}_{s,i}^{(0)}$ represents the coordinates of the ith surface mesh point in the baseline (undeformed) configuration and $\boldsymbol{x}_{s,i}$ are the coordinates of this point after deformation. The matrix $\boldsymbol{M}_i$ and the vector $\boldsymbol{b}_i$ define, respectively, the rotation and translation applied to the surface mesh point i.

To compute the rotation matrix $\boldsymbol{M}_i$, we first assign a normal vector $\boldsymbol{n}_i$ to each surface point. This is done by taking a weighted average of the normal vectors ($\boldsymbol{n}_{\text{face}}$) of the face elements surrounding point i, using the element areas (A_{face}) as weights:

$$\boldsymbol{n}_i = \frac{\sum_{\text{faces}} \dfrac{A_{\text{face}}}{c_{\text{face}}} \boldsymbol{n}_{\text{face}}}{\sum\limits_{\text{faces}} \dfrac{A_{\text{face}}}{c_{\text{face}}}} \tag{39}$$

where c_{face} is the number of points that share the current face element ($c_{\text{face}} = 4$ for structured surface meshes using quadrilateral elements, for example). This process is performed for both the undeformed and deformed surface meshes, yielding the point normal vectors $\boldsymbol{n}_i^{(0)}$ and $\boldsymbol{n}_i$, respectively. We define $\boldsymbol{M}_i$ as the rotation matrix that transforms $\boldsymbol{n}_i^{(0)}$ into $\boldsymbol{n}_i$, which can be computed using the Rodrigues' rotation formula.

Once $\boldsymbol{M}_i$ is computed, we rearrange Eq. (38) to determine $\boldsymbol{b}_i$ as

$$\boldsymbol{b}_i = \boldsymbol{x}_{s,i} - \boldsymbol{M}_i \boldsymbol{x}_{s,i}^{(0)} \tag{40}$$

We repeat this process for every surface point i to compute their corresponding $\boldsymbol{M}_i$ and $\boldsymbol{b}_i$.

Next, we need to propagate the deformation of surface mesh to volume mesh. The deformation of surface mesh point i induces a deformation $\boldsymbol{s}_i$ at a volume point $\boldsymbol{x}^{(0)}$:

$$\boldsymbol{s}_i(\boldsymbol{x}^{(0)}) = \boldsymbol{M}_i \cdot \boldsymbol{x}^{(0)} + \boldsymbol{b}_i - \boldsymbol{x}^{(0)} \tag{41}$$

where $\boldsymbol{x}^{(0)}$ denotes the volume mesh coordinates of the baseline, undeformed mesh. For the volume point $\boldsymbol{x}^{(0)}$, its deformation $\Delta \boldsymbol{x}$ is

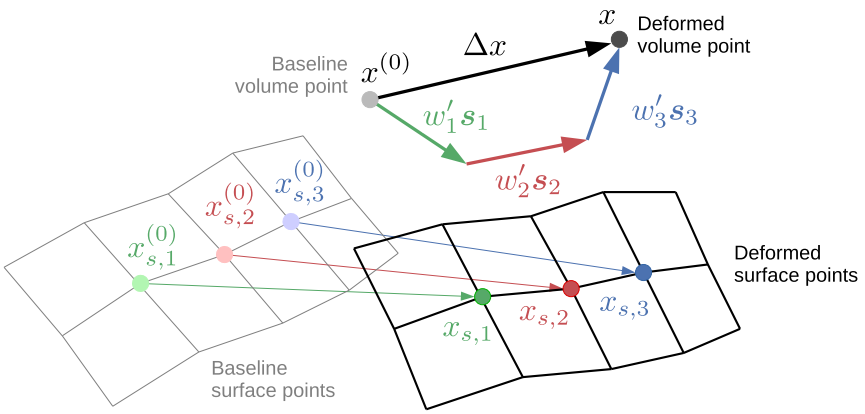


Fig. 9 The volume point deformation is a weighted average of the deformations induced by surface points, where w_i' are the weights normalized by their sum.

then computed as the weighted average of deformations $\boldsymbol{s}_i$ induced by all surface mesh points (see Fig. 9):

$$\Delta \boldsymbol{x}(\boldsymbol{x}^{(0)}) = \boldsymbol{x} - \boldsymbol{x}^{(0)} = \frac{\sum_i w_i(\boldsymbol{x}^{(0)}) \cdot \boldsymbol{s}_i(\boldsymbol{x}^{(0)})}{\sum_i w_i(\boldsymbol{x}^{(0)})} \tag{42}$$

where $\boldsymbol{x}$ is the deformed position of the baseline volume point $\boldsymbol{x}^{(0)}$. The weights w_i used in the averaging process are inversely proportional to the distance between the volume point and the corresponding surface point:

$$w_i(\boldsymbol{x}^{(0)}) = A_i^{(0)} \left[\left(\frac{L_{\text{def}}}{|\boldsymbol{x}^{(0)} - \boldsymbol{x}_{s,i}^{(0)}|} \right)^a + \left(\frac{\alpha L_{\text{def}}}{|\boldsymbol{x}^{(0)} - \boldsymbol{x}_{s,i}^{(0)}|} \right)^b \right] \tag{43}$$

We divide the area of each surface element equally among its corner points to determine the nodal areas, $A_i^{(0)}$, of the undeformed surface cells. The parameter L_{def} controls how far surface deformations propagate in the volume and is usually set to the maximum distance between the surface mesh centroid and any surface mesh point. The factor α is used in conjunction with the exponents a and b to define the length of a near-field deformation region, where deformations of the nearby surface modes become more relevant (by having higher weights) than distant ones in the averaging process. The typical values for these are $\alpha = 0.25$, $a = 3$, and $b = 5$. The weights w_i remain constant throughout the optimization because they depend only on the undeformed geometry.

One advantage of the IDW method is that only surface connectivity information is necessary to compute the nodal normals, $\boldsymbol{n}_i$, and nodal areas, A_i. Since volume points are treated as a cloud of points, this method can be used for both structured and unstructured meshes.

Summing the contributions of every surface patch over each volume point becomes costly for three-dimensional meshes [8]. We reduce the computational cost by using kd-trees [41,42] to approximate the numerator and the denominator of Eq. (42). The central idea is that, instead of computing the individual contribution of surface point deformation to the volume mesh points, we compute the aggregate contribution from a group of surface points within a kd-tree node. The details are as follows.

In our implementation, surface mesh points can be assigned to multiple kd-tree nodes. All surface mesh points are initially assigned to the root node of the kd-tree. This root node has two child nodes, each containing half of the surface points, which are split based on the coordinate (x, y, or z) that has the widest range. For instance, if the y coordinate has the largest variation among all coordinates, the half of the surface points with lowest values of y are assigned to one child node, while the surface points with the largest values of y are assigned to the other node. Then, each child node becomes a parent of two other child nodes that receives half of the surface points assigned to their parent. The coordinate used to subdivide the points is the one that has the largest spread, considering just the points assigned to the parent node. This process is repeated until a node has fewer points than a user-specified threshold, which is usually eight points, thus becoming a leaf of the tree. Algorithm 1 summarizes the kd-tree building process.

Figure 10 illustrates the structure of a kd-tree in a bidimensional situation. Initially, all surface mesh points (black dots) are assigned to the root node. Since these points are more widely distributed along the x coordinate, this first cut (cut A) assigns the half of the surface points with smallest x values to the first child node (node 1) and the other half of points with the greatest x values to the second child node (node 2). The points within node 1 are still more widely distributed along the x coordinate, so another cut (cut B) splits these points based on their x values. Conversely, the y coordinate has a wider range of values in node 2 compared with the x coordinate. Therefore, the third cut (cut C) uses y values to further subdivide points in child nodes. The process can be repeated if finer subdivisions are necessary.

We condense the individual properties of the surface points, such as areas and normals, into nodal properties. Therefore, instead of computing the contributions of multiple surface points to the mesh deformation, we use the single contribution from a kd-tree node containing these surface points. We take the condensed area A_i of a

Algorithm 1: Build kd-tree

```
procedure BuildNode(nodeID)                                   ▷ Define procedure for tree traversal
    if number of surface points assigned to this node < user-specified threshold then
        Set this node as a leaf node
    else
        Check which coordinate (x, y, or z) has the largest spread among the assigned surface points
        Sort the surface points based on the selected coordinate
        Create childNode1 and assign the first half of surface points to it
        Create childNode2 and assign the second half of surface points to it
        Call BuildNode(childNode1)
        Call BuildNode(childNode2)
Assign all surface points to the root node
Call BuildNode(rootNode)                                      ▷ Start tree building from the root node
```

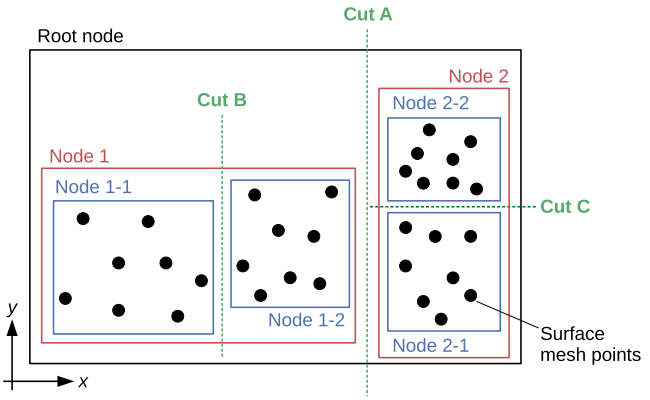


Fig. 10 Illustration of the kd-tree building process.

leaf node as the sum of areas of the surface points assigned to it, while the condensed values of $\boldsymbol{M}_i$, $\boldsymbol{b}_i$, and $\boldsymbol{x}_{s,i}^{(0)}$ are computed by averaging values of the surface points.

Once we have the condensed properties of the leaf nodes, we can obtain the properties of the parent nodes. The area of a parent node is the sum of its children areas, and $\boldsymbol{M}_i$, $\boldsymbol{b}_i$, and $\boldsymbol{x}_{s,i}^{(0)}$ are the average of its children values. Only the $\boldsymbol{M}_i$ and $\boldsymbol{b}_i$ values of the tree nodes need to be updated after each surface modification.

If a volume mesh point is too close to the surface points belonging to a given tree node, the use of condensed nodal properties to compute Eq. (42) leads to greater approximation errors. Thus, we need to define an acceptable range beyond which the use of condensed properties is suitable.

After the tree building process, we take each node of the tree and create a series of 20 dodecahedrons surrounding them. The radii of these dodecahedrons vary from 2 to 640 times the radius of the sphere that encloses all surface points of the given tree node. We compute the denominator of Eq. (42) at the dodecahedron points using the condensed properties of the tree node and compare this value against the exact denominator computed with the contributions of each surface point belonging to the tree node. This allows us to track how the average error in the denominator changes with the distance from the tree node centroid.

When evaluating the contribution of a tree node to the denominator of a volume mesh point, we compute the distance between this point and the tree node centroid. If this distance results in a denominator error lower than a user-provided threshold, we use the condensed properties of the node to compute the contribution of the contained surface points. Otherwise, we go deeper into the tree by recursively applying this denominator evaluation process for both child nodes, spawning additional paths in the tree traversal process. If we reach a leaf node of the tree, we compute the exact denominator contribution using the all the surface points of the tree.

The process described in the previous paragraph starts at the root node of the kd-tree to compute the denominator of each volume mesh point. Some recursive evaluations stop at intermediate tree nodes, where the contributions of faraway surface points with respect to the volume mesh point are given by the condensed properties of the node, while contributions of nearby surface points are computed exactly when the recursion reaches the leaf nodes. The tree traversal process is detailed in Algorithm 2.

Figure 11 shows the main structure of the mesh deformation code, where we highlight which subroutines are used only in the initialization procedure. The dots $(\cdot)$ indicate which attributes of the kd-tree are

Algorithm 2: Displacement computation for volume mesh point with kd-tree traversal

$\boldsymbol{x}^{(0)} \leftarrow$ Coordinates of undeformed volume mesh point
$D \leftarrow 0$ ▷ Initialize denominator value
$\boldsymbol{N} \leftarrow \boldsymbol{0}$ ▷ Initialize numerator value
procedure CHECKNODE($\boldsymbol{x}^{(0)}$, *nodeID*, D, $\boldsymbol{N}$) ▷ Define recursive procedure for tree traversal
  if *nodeID* is a leaf node **then**
    for every surface point of the leaf node **do**
      Compute w_i and $\boldsymbol{s}_i$ using the surface point properties
      $D \leftarrow D + w_i$
      $\boldsymbol{N} \leftarrow \boldsymbol{N} + w_i \boldsymbol{s}_i$
  else
    $d \leftarrow$ Distance between $\boldsymbol{x}^{(0)}$ and the node centroid
    $error \leftarrow$ Interpolated w_i error for a dodecahedron of radius d around the tree node
    if $error > toleratedError$ **then**
      call CheckNode($\boldsymbol{x}^{(0)}$, *childNode1*, D, $\boldsymbol{N}$)
      call CheckNode($\boldsymbol{x}^{(0)}$, *childNode2*, D, $\boldsymbol{N}$)
    else
      Compute w_i and $\boldsymbol{s}_i$ using the condensed properties of the tree node
      $D \leftarrow D + w_i$
      $\boldsymbol{N} \leftarrow \boldsymbol{N} + w_i \boldsymbol{s}_i$
call CheckNode($\boldsymbol{x}^{(0)}$, *rootNode*, D, $\boldsymbol{N}$) ▷ Start traversal at the root node to accumulate D and $\boldsymbol{N}$
$\Delta\boldsymbol{x} \leftarrow \boldsymbol{N}/D$ ▷ Compute volume mesh point displacement
$\boldsymbol{x} = \boldsymbol{x}^{(0)} + \Delta\boldsymbol{x}$ ▷ Compute the position of the deformed volume mesh point

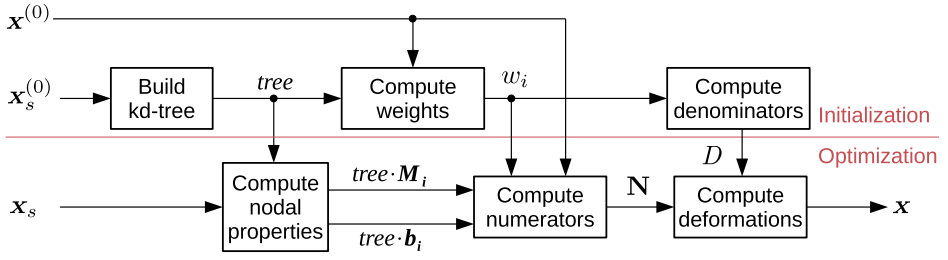


Fig. 11 Components of the mesh deformation code used during the initialization and optimization steps.

updated after each step. The variables D and N represent the denominator and numerator values of Eq. (42), respectively. "Build kd-tree" corresponds to Algorithm 1 and "Compute weights," "Compute denominators," "Compute numerators," and "Compute deformations" correspond to Algorithm 2. The performance of the mesh deformation will be evaluated in Sec. IV.B.

We need efficient ways for computing derivatives across the mesh deformation module, since it is an integral component of the optimization framework. Once again, we use reverse AD via Tapenade to obtain differentiated versions of the mesh deformation code.

We can selectively apply the differentiation engine to certain components of the mesh deformation tool to obtain a more efficient differentiated code [43], since components and parameters on the initialization side of Fig. 11 remain constant throughout the optimization. For instance, the kd-tree building process depends only on the undeformed surface mesh. Consequently, there is no need to differentiate the tree-building component. Figure 12 shows the specific code components that are differentiated by Tapenade to build the reverse AD version of the mesh deformation code.

The derivative seeds employed by the differentiated code are related to the derivatives of the volume points coordinates with respect to the surface point coordinates as follows:

$$\overline{\boldsymbol{x}}_s = \left[\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{x}_s}\right]^T \cdot \overline{\boldsymbol{x}} \tag{44}$$

The overall reverse AD chain of the optimization framework dictates which volume point derivative seeds ($\overline{\boldsymbol{x}}$) are used to propagate derivatives across the mesh deformation module.

An ASO framework has to compute aerodynamic functions of interest $\boldsymbol{f}$, such as lift and drag, based on a set of design variables $\boldsymbol{p}$, usually subdivided into shape design variables $\boldsymbol{p}_{\text{shape}}$ and flow condition design variables $\boldsymbol{p}_{\text{flow}}$ (such as angle of attack or Mach number). Such frameworks generally include a geometry manipulation tool that receives a set of user-defined design variables and apply the corresponding modifications to the surface meshes ($\boldsymbol{x}_s$), a mesh deformation tool that uses the modified surface meshes to propagate deformations to the volume mesh ($\boldsymbol{x}$), and a CFD solver that takes the deformed meshes and finally computes the desired functions of interest (upper section of Fig. 13).

As mentioned in Sec. II.B, gradient-based optimization is advantageous for ASO problems because it can handle a large number of design variables. Other components of the framework besides the mesh deformation tool can be differentiated to yield a reverse AD chain (lower section of Fig. 13). The CFD solver can be differentiated by selectively applying reverse AD to its subroutines to avoid memory bottlenecks [43,44]. More specifically, we need to differentiate 1) the subroutine that computes the CFD residuals ($\boldsymbol{r}$) based on the volume mesh points ($\boldsymbol{x}$), flow state variables ($\boldsymbol{w}$), and flow conditions ($\boldsymbol{p}_{\text{flow}}$), and 2) the subroutine that computes the functions of interest ($\boldsymbol{f}$), which is also based on the current mesh points and flow state. All intermediate subroutines called by these two subroutines, such as boundary conditions and cell metrics subroutines, are also differentiated.

The reverse AD version of the framework takes the seeds of residuals and functions of interest ($\overline{\boldsymbol{r}}$ and $\overline{\boldsymbol{f}}$) as inputs and returns the derivative of the design and state variables ($\overline{\boldsymbol{p}}$ and $\overline{w}$):

$$\overline{\boldsymbol{p}} = \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^T \cdot \overline{\boldsymbol{r}} + \left[\frac{\partial f}{\partial \boldsymbol{p}}\right]^T \cdot \overline{f} \tag{45a}$$

$$\overline{\boldsymbol{w}} = \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{w}}\right]^T \cdot \overline{\boldsymbol{r}} + \left[\frac{\partial f}{\partial \boldsymbol{w}}\right]^T \cdot \overline{f} \tag{45b}$$

The total derivative of the function f with respect to the design variables and the corresponding discrete adjoint equation that determines the adjoint variables $\boldsymbol{\psi}$ are

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{p}} = \frac{\partial f}{\partial \boldsymbol{p}} + \boldsymbol{\psi}^T \cdot \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}} \tag{46a}$$

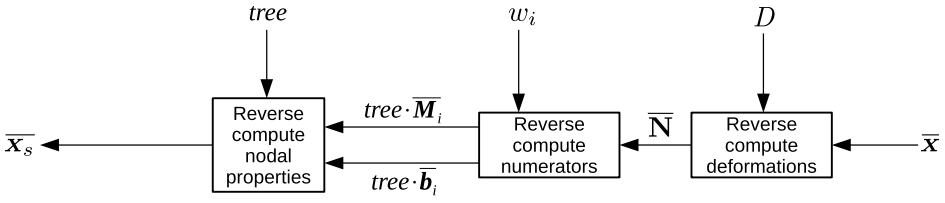


Fig. 12 Reverse differentiation version of the mesh deformation code.

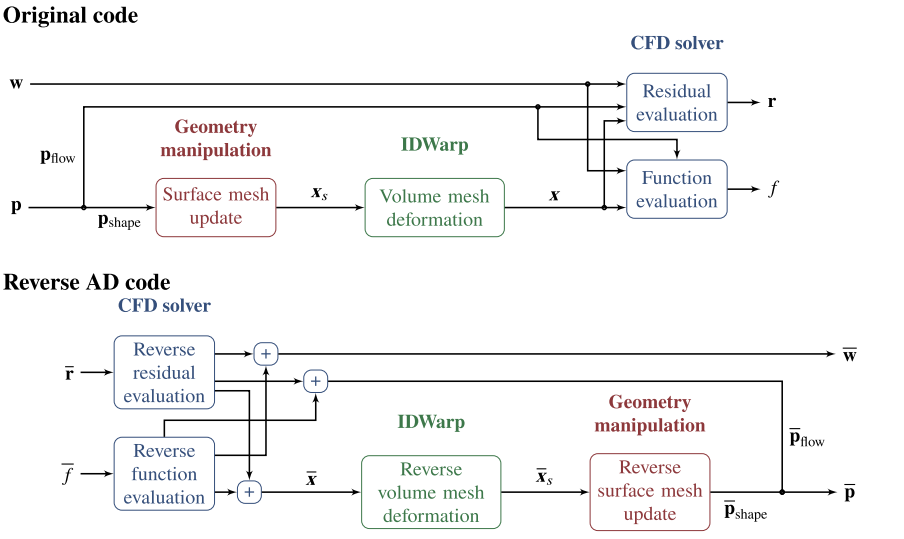


Fig. 13 Aerodynamic analysis framework and its reverse AD counterpart. The plus signs indicate that the derivative seed should be accumulated from multiple sources. The reverse IDWarp component is detailed in Fig 12.

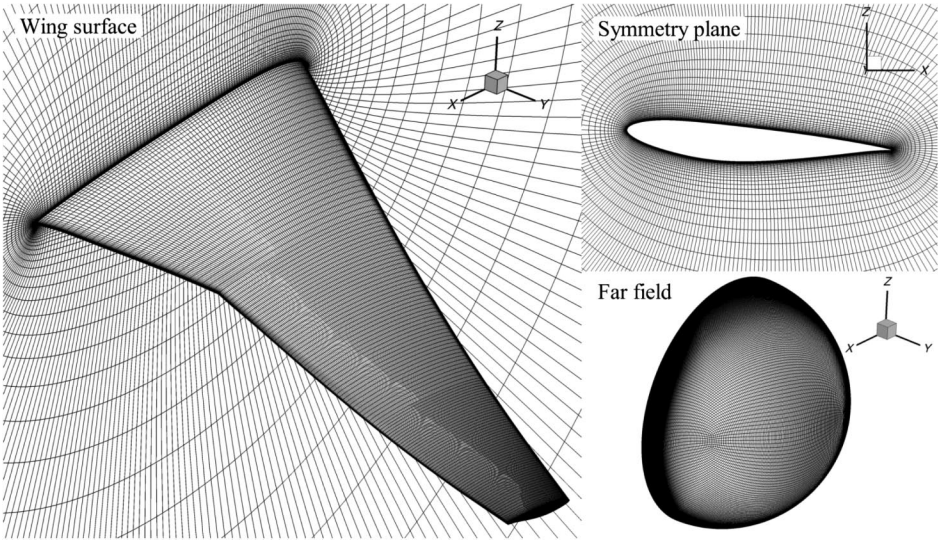


Fig. 14 Multiblock structured mesh generated by pyHyp.

$$\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{w}}\right]^T \cdot \boldsymbol{\psi} + \left[\frac{\partial f}{\partial \boldsymbol{w}}\right]^T = 0 \tag{46b}$$

Due to the similarities between Eq. (45) and Eq. (46), we can choose appropriate values of $\overline{\boldsymbol{r}}$ and $\overline{f}$ to recreate the terms to solve the adjoint equation and to compute the total derivative of the function of interest [25,43]. This combined use of reverse AD and the analytic adjoint method avoids the reverse differentiation of the iterative CFD solution process, which would lead to prohibitive memory usage [43,44]. Kenway et al. [43] also present an extensive verification of the accuracy of the derivatives computed using this methodology and the associated memory usage, which is about twice the memory used for a normal flow solution.

IV. Results and Discussion

In this section, we evaluate the performance of the proposed mesh generation and deformation approaches in terms of speed, scalability, and robustness. We also demonstrate a wide range of ASO applications that use the mesh deformation module.

A. Mesh Generation

We first evaluate the performance of the proposed mesh generation methods. To this end, we use the Common Research Model (CRM) wing [45] as the baseline geometry, as shown in Fig. 14. We performed single- and multi-point ASO using this model in our previous studies [46,47]. Here we use the CRM wing L1 mesh, which has 3.56 million cells. As mentioned above, we first generate the multiblock wing surface mesh manually using ICEM-CFD [26]. Then, we extrude the volume mesh using pyHyp, whose hyperbolic mesh extrusion approach ensures a good mesh orthogonality near the wing surface, as shown in Fig. 14. The histogram of mesh quality, quantified as the mesh determinant, is shown in Fig. 15. This metric computes the normalized determinant of the Jacobian for a mesh cell. A perfect hexahedral cube has a determinant of 1.0, whereas a zero determinant would correspond to an inverted cube whose volume is negative [26]. Most of CFD solvers can run with a mesh determinant larger than 0.01, while good-quality meshes typically have determinants larger than 0.2. As shown in Fig. 15, 99% of mesh cells have mesh quality larger than 0.6, with a mean mesh quality of 0.83 and a minimum of 0.23.

We generate the above mesh using the Stampede 2 supercomputer.[‡‡] The Skylake nodes are equipped with Intel Xeon Platinum

[‡‡]https://www.tacc.utexas.edu/systems/stampede2.

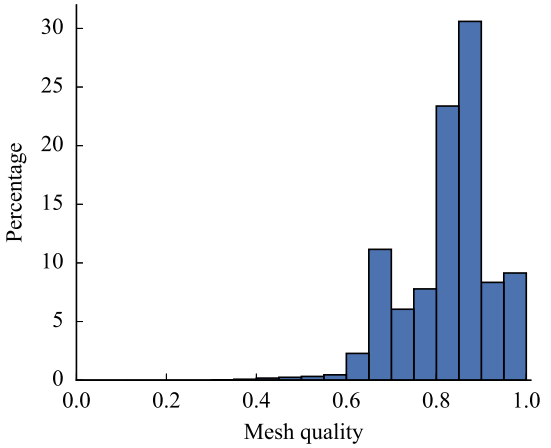


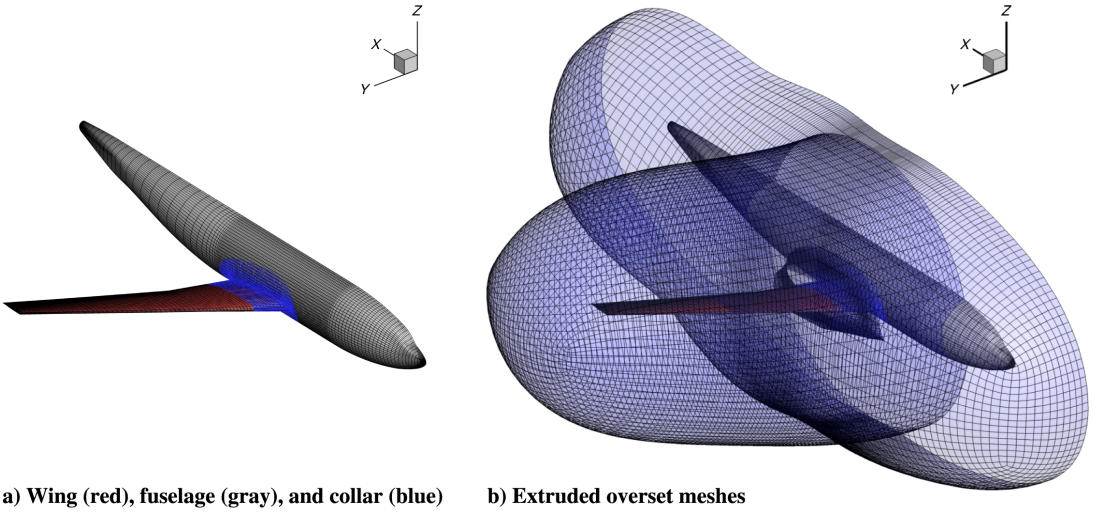
Fig. 15 pyHyp generates high-quality structured meshes with a mean mesh quality of 0.83; the minimum is 0.23.

Table 1 pyHyp generates high-quality structured mesh with up to 100 million cells using 256 CPU cores for the CRM wing case

Cells (million)	Nodes	CPU cores	Runtime (s)	Memory (GB)	Minimum quality	Mean quality
0.45	1	1	10.8	0.1	0.19	0.77
3.56	1	8	34.2	1.0	0.23	0.83
28.48	2	64	90.3	7.8	0.19	0.87
113.92	8	256	376.4	32.7	0.19	0.86

Table 2 pyHyp exhibits reasonable strong scalability up to 64 CPU cores for the CRM wing case with 3.56 million cells

Nodes	CPU cores	Runtime (s)
1	4	76.9
1	8	34.2
2	16	25.8
4	32	16.5
8	64	12.1



a) Wing (red), fuselage (gray), and collar (blue) surface meshes **b) Extruded overset meshes**

Fig. 16 The automatic collar mesh generation module (pySurf) enables the meshing of complex geometries using the overset approach.

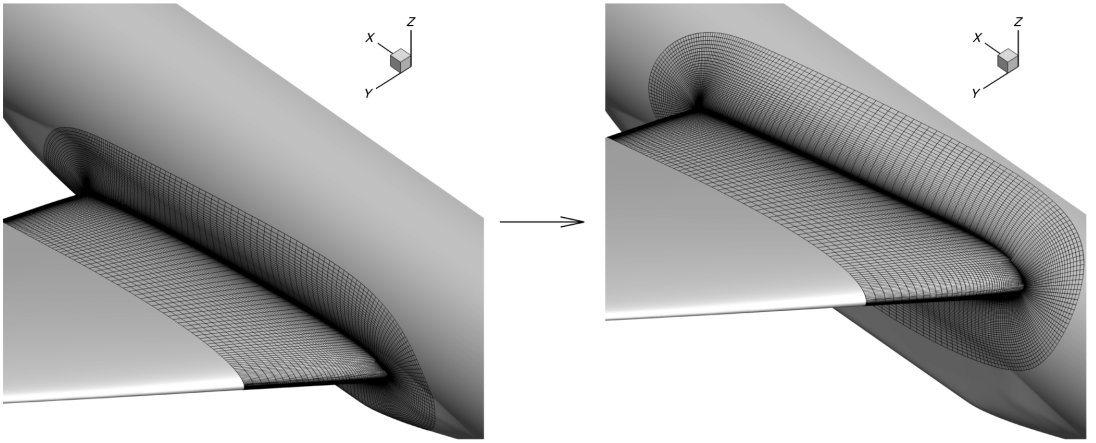


Fig. 17 The wing intersection of the CRM configuration can be moved up using pySurf, which recomputes the intersection and regenerates collar meshes automatically.

8160 CPUs running at 2.1 GHz, where each node has 48 CPU cores and 196 GB of memory. The mesh generation takes 34.2 s and 1.0 GB memory using 8 CPU cores. To evaluate the weak scalability of pyHyp, we generate three additional meshes with 0.45, 28.48, and 113.92 million cells, as shown in Table 1. In addition, we evaluate the strong scalability for the 3.56 million cell case, as shown in Table 2. Our proposed approach generates high-quality structured meshes with up to 100 million cells using 256 CPU cores (see detailed discussion on mesh quality in Sec. IV.B). The speed and scalability of the mesh generation is acceptable for practical ASO because we need to generate the baseline mesh only once.

In addition to mesh generation for a single geometry component, we demonstrate the overset mesh generation capability using pySurf. The model is the DLR-F6 wing–fuselage configuration [48], shown in Fig. 16. We use ICEM-CFD to generate the structured surface meshes (gray and red meshes of Fig. 16a) and triangulated surfaces for the wing and fuselage components. Then, pySurf generates the surface collar mesh at the wing–fuselage intersection (blue mesh of Fig. 16a). Finally, we use pyHyp to extrude the volume meshes and construct the overset mesh (Fig. 16b).

We also use pySurf to automatically regenerate the wing–body intersection curve of the CRM as we move the wing up relative to the fuselage, as shown in Fig. 17. Both wing and fuselage are triangulated surfaces, and pySurf recomputes the intersection curve and the collar mesh for any wing position. The generation of collar meshes for intersections has been discussed in previous work [49].

B. Mesh Deformation

We now demonstrate the mesh deformation capability using IDWarp. The baseline mesh is the CRM wing L1 mesh, which has 3.56 million cells. We change the surface geometry (and therefore the surface mesh) using the free-form deformation (FFD) approach, implemented in the open-source geometry engine pyGeo.[§§] We set up three global design variables (twist, sweep, and span) to change the planform of the CRM wing. For the twist design variable, we set two sections of FFD points, one at the root and one at the tip. We then rotate the FFD points at the tip, while keeping the root FFD points fixed. For the sweep and span design variables, we move all the FFD points in the x (streamwise) and y (spanwise) directions, respectively.

Figure 18 shows the extreme deformations achieved for twist, sweep, and span. The values for these deformations are listed in Table 3. We consider a mesh deformation to be valid if the mesh quality remains larger than 0.01, as mentioned before. Despite the relative large mesh deformation, the mesh orthogonality near the

[§§]https://github.com/mdolab/pygeo.

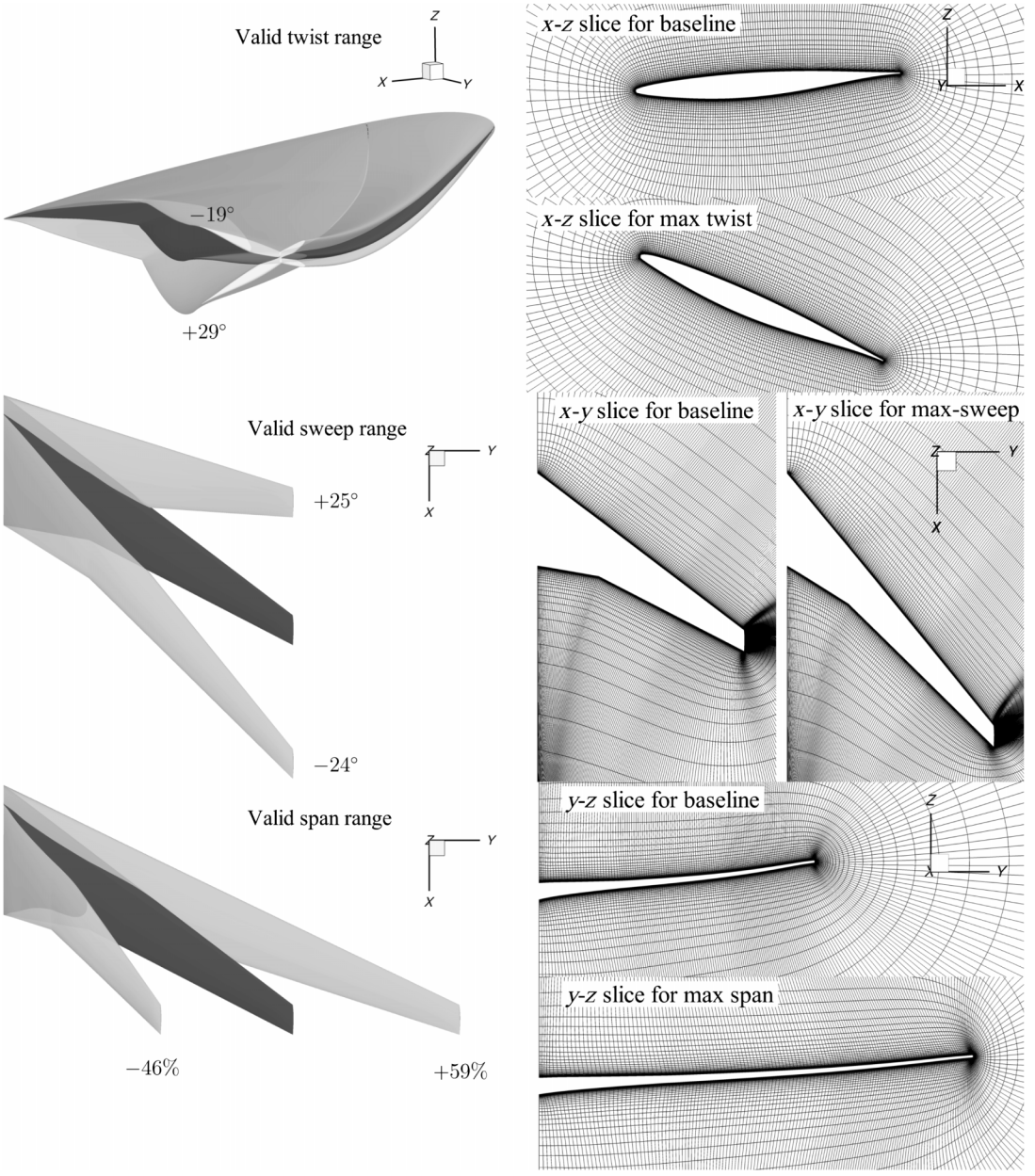


Fig. 18 The robust mesh deformation algorithm in IDWarp enables a wide range of valid mesh deformations for structured hexahedral meshes.

wall is well preserved. The range of valid mesh deformation for twist, sweep, and span is sufficient for practical ASO, which demonstrates the robustness of our mesh deformation approach.

The performance of the mesh deformation algorithm is summarized in Table 4. The mesh deformation and the corresponding mesh derivative computation are fully parallelized and their computational cost is only about 0.1% of the CFD runtime.

To further demonstrate the flexibility of the proposed mesh deformation approach, we generate two types of unstructured meshes for the same CRM wing. The first one is an unstructured hexahedral mesh with 3.54 million cells generated by the snappyHexMesh utility from OpenFOAM [50,51], and the second mesh is an unstructured triangular mesh with 3.23 million cells generated in ICEM-CFD. We then set the same design variables (twist, sweep,

Table 3 The robust mesh deformation algorithm in IDWarp yields valid meshes for a wide range of mesh deformations

Mesh type	Twist	Sweep	Span
Structured Hex	−19° to 29°	−24° to 25°	−46% to 59%
Unstructured Hex	−35° to 36°	−17° to 20°	−51% to 40%
Unstructured Tri	−23° to 25°	−15° to 17°	−39% to 35%

Table 4 The computational cost of the mesh deformation and derivative computation is only about 0.1% of the CFD runtime for the CRM wing structured meshes

CPU cores	CFD (s)	Mesh deformation (s)	Deformation derivative (s)
12	1062.6	1.6 (0.15%)	1.9 (0.18%)
24	729.0	0.9 (0.12%)	1.1 (0.15%)
48	328.2	0.5 (0.15%)	0.7 (0.21%)
96	238.6	0.3 (0.12%)	0.4 (0.17%)

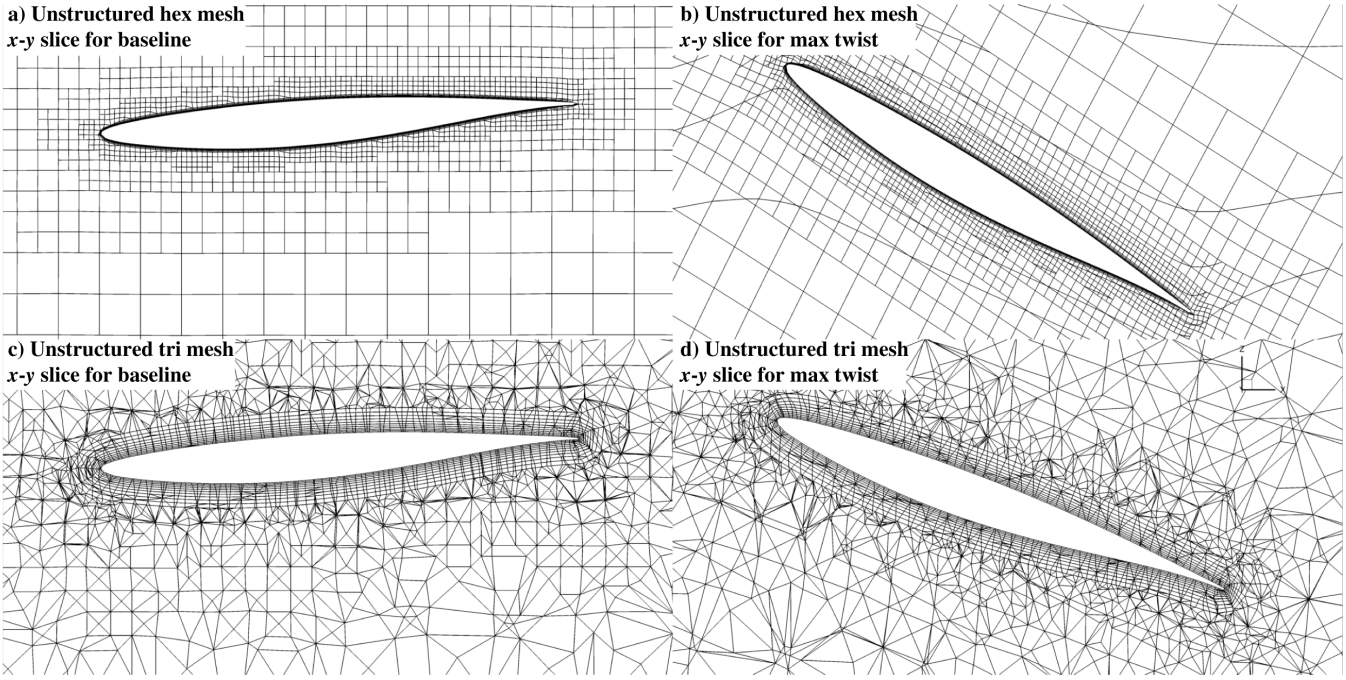


Fig. 19 The proposed mesh deformation approach can also be used for unstructured meshes.

and span) to deform these unstructured meshes. The maximum twist deformation achieved with a valid mesh is illustrated in Fig. 19, and the range of valid deformation is summarized in Table 3. For the triangular mesh, we use the mesh-determinant metric to quantify the mesh quality, which is the same indicator used for the structured mesh. However, for the unstructured hexahedral mesh, we use the minimum mesh angle as the mesh quality, to be consistent with that used in OpenFOAM. This metric measures the angle between the face normal and the vector that connects two adjacent cell centers. A mesh angle of 90 deg indicates a perfectly orthogonal cell, whereas an extremely skewed cell has a mesh angle close to zero degrees. We consider the valid deformation when the minimum angle is larger than 20 deg, which is the default threshold in OpenFOAM. Similar to what we observed for the structured mesh, the near-wall mesh orthogonality is well preserved, despite the relatively large deformation (Fig. 19). In addition, the valid mesh deformation range of the unstructured meshes is generally smaller than that of the structured meshes (Table 3); nevertheless, it is more than sufficient for practical ASO.

Figure 20 shows the impact of the mesh deformation on mesh quality. The design variable values (twist, span, and sweep) are relative to the nominal geometry. There is a steep decrease in the mesh quality as the twist increases or decreases relative to the nominal value. The mesh quality is much less sensitive to the sweep and span, especially for the structured hexahedral mesh. In some cases, the mesh quality remains almost unchanged, such as when decreasing the sweep and increasing the span for the structured mesh.

Finally, we verify the accuracy of our mesh deformation derivative by comparing values computed by AD, finite-difference (FD) approximations, and complex-step (CS) method [52]. We use a swept wing with the RAE2822 profile as the test case. For the FD approach, we apply random perturbations of varying step sizes to the 1239 surface points of the wing, deform the volume mesh using the IDW algorithm, compute the displacement of the corresponding 37,615 volume mesh points, then use the forward difference scheme to compute the mesh deformation derivatives. For the AD approach, we compute the derivative using the approach elaborated on in Sec. III. Then we compute the maximum difference between the FD and AD approaches, as shown in Fig. 21. Similar comparison is also shown for the CS derivatives. As expected, we see a tradeoff between truncation error and subtractive cancellation error for the FD derivatives, while the discrepancy between AD and CS remains within machine precision for all step sizes [52]. This verifies the accuracy of mesh deformation derivatives using the AD approach.

As mentioned before, the proposed mesh deformation approach has been implemented in IDWarp, incorporated into a high-fidelity gradient-based optimization framework MACH-Aero, and performed ASO for a wide range of applications in previously published work, including aircraft, airframe–propulsion integration, wind turbines, compressors, cars, and internal cooling channels, as summarized in Table 5.

The robust deformation algorithm allows a large mesh deformation, which increases the freedom to more efficiently explore the design space. For example, we performed ASO to turn a circle to an airfoil [53] using ADflow [23] in MACH-Aero. In addition, a structural solver TACS [54] was integrated into the MACH framework to perform high-fidelity aerostructural optimization, where the robust deformation in IDWarp handles highly flexible wings with planform changes [36,55–58]. Furthermore, the MACH-Aero framework has been coupled with OpenMDAO [59] to handle aeropropulsive optimizations [60–63]. In addition to structured meshes, the flexibility of IDWarp enables us to handle unstructured mesh deformation [24,64] using DAFoam [24] in the MACH-Aero framework.

V. Conclusions

Efficient mesh generation and deformation approaches are developed and discussed in this paper. A semi-automatic approach is developed to generate high-quality structured meshes for single geometry components. This approach starts with the manual generation of a structured surface mesh and then extrudes the volume mesh automatically by solving a set of hyperbolic equations. In addition, an automatic approach is proposed to generate surface meshes for intersections (collar meshes) between multiple geometry components. The mesh generation method scales well for cases with up to 100 million cells using 256 CPU cores, and automatically handles collar meshes for wing–fuselage intersections.

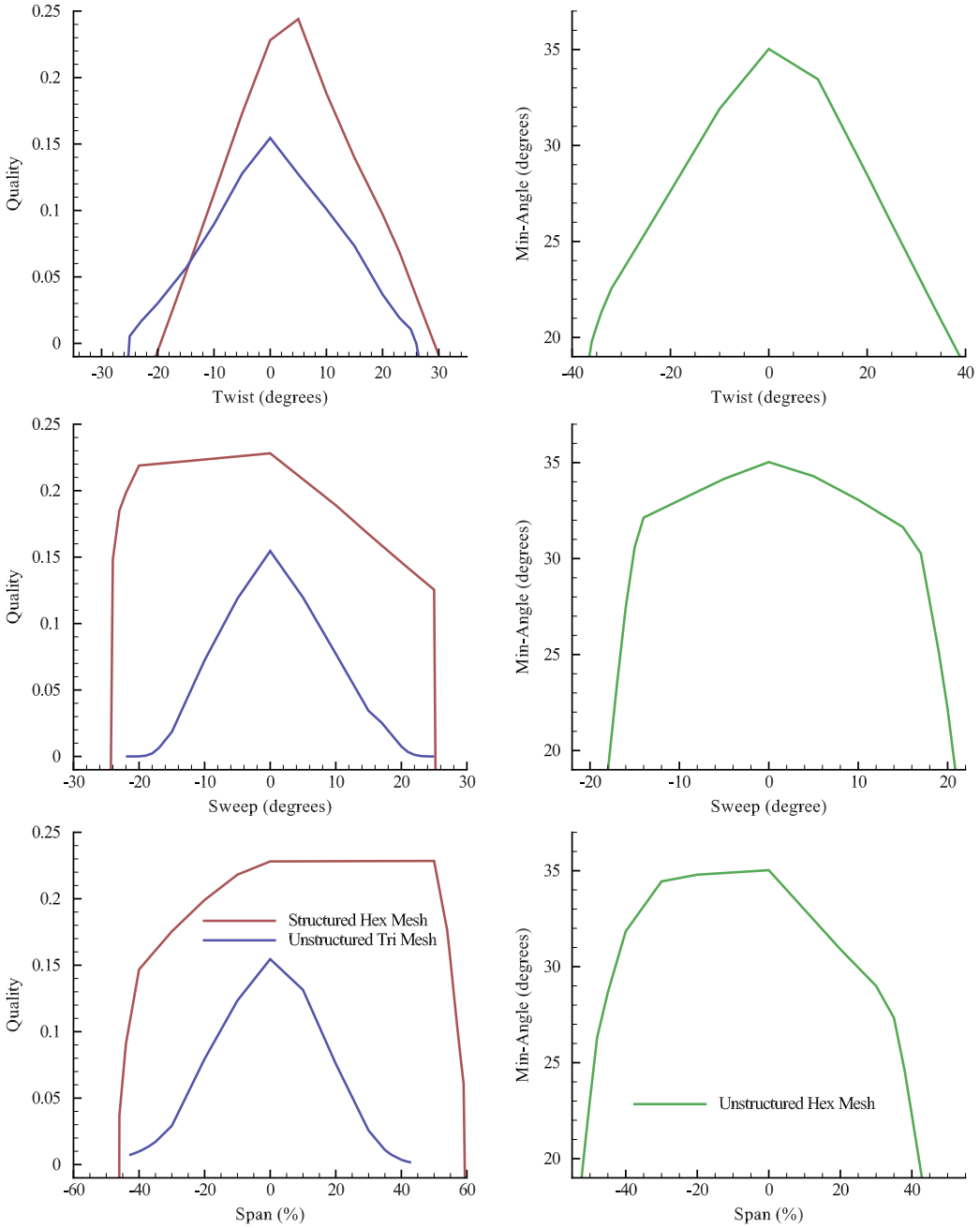


Fig. 20 Variation of mesh quality with respect to deformation. The meshes remain valid for a wide range of deformations. The design variable values are relative to the nominal geometry.

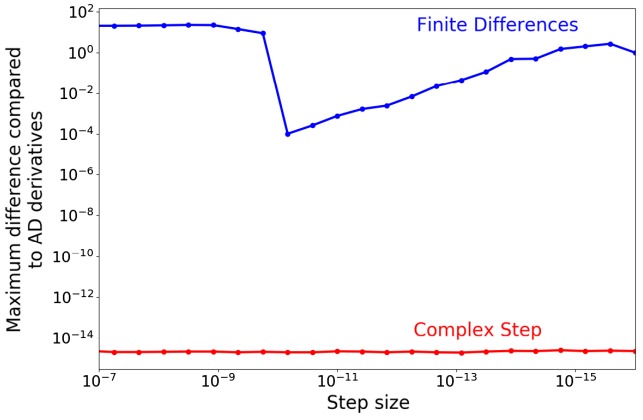


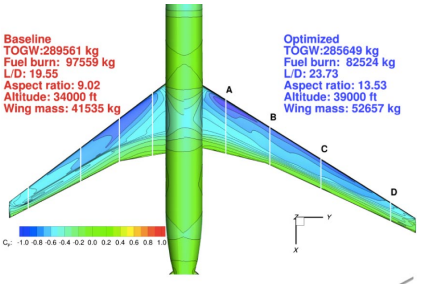
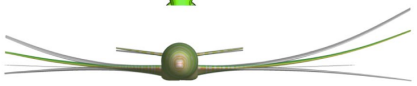
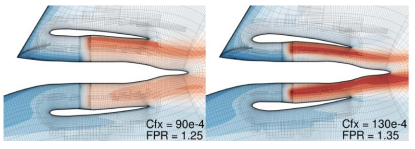
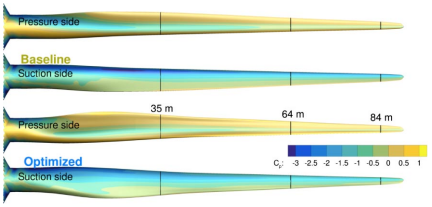
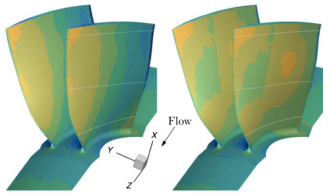
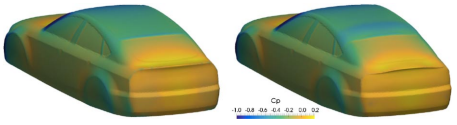
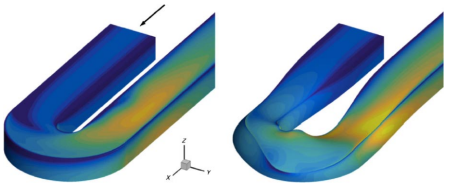
Fig. 21 Comparison of IDWarp mesh deformation derivatives computed by finite-difference, complex-step, and AD methods.

In addition to mesh generation, a mesh deformation method is developed that uses an inverse distance weighting algorithm to deform the volume mesh. A critical feature of this approach is that no volume mesh connectivity information is required. Therefore, this approach is applicable to both structured and unstructured mesh types. It also allows a broad range of valid mesh deformations, which increases the freedom to explore the design space in ASO. In addition, the derivatives of the mesh deformation process are efficiently computed using reverse AD. Both the mesh deformation and the derivative computation require only 0.1% of total CFD runtime.

The mesh generation and deformation approaches have been implemented in pyHyp, pySurf, and IDWarp packages. The proposed approaches have been proved useful in general ASO applications, including aircraft, turbomachinery, and ground vehicle design.

Future work may include adding new features to pySurf to handle simultaneous intersections of three or more components. Also, investigations on the underlying mathematical properties of the mesh generation PDEs may provide insights about this

Table 5 Summary of design optimization applications where the proposed mesh deformation has been used

Examples of mesh deformation	Topics	References
	Aerodynamic optimization of airfoils	[53,65–67]
	Aerodynamic and aerostructural optimization of aircraft configurations	[46,55–58,68–73]
 	Aerodynamic optimization for airframe–propulsion integration	[59–63]
	Aerodynamic optimization of wind turbines	[74,75]
	Aerodynamic optimization of axial compressors	[24]
	Aerodynamic optimization of ground vehicles	[64]
	Aerothermal optimization of turbine internal cooling channels	[76]

methodology's robustness and indicate valid ranges for the coefficients used throughout the process, further increasing its degree of automation.

Acknowledgments

The first author acknowledges funding from the Brazilian Air Force. The authors also thank Shamsheer Chauhan and Yingqian Liao for their helpful comments toward improving the paper. The computations were done in the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation Grant number ACI-1548562.

References

[1] Mavriplis, D., "Unstructured Grid Techniques," *Annual Review of Fluid Mechanics*, Vol. 29, No. 1, 1997, pp. 473–514.

[2] Kenway, G. K. W., Secco, N., Martins, J. R. R. A., Mishra, A., and Duraisamy, K., "An Efficient Parallel Overset Method for Aerodynamic Shape Optimization," *Proceedings of the 58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech Forum*, AIAA Paper 2017-0357, 2017. https://doi.org/10.2514/6.2017-0357

[3] Chan, W. M., and Steger, J. L., "Enhancements of a Three-Dimensional Hyperbolic Grid Generation Scheme," *Applied Mathematics and Computation*, Vol. 51, Nos. 2–3, 1992, pp. 181–205. https://doi.org/10.1016/0096-3003(92)90073-A

[4] Chan, W., Gomez, R., Rogers, S., and Buning, P., "Best Practices in Overset Grid Generation," *32nd AIAA Fluid Dynamics Conference and Exhibit*, AIAA Paper 2002-3191, 2002. https://doi.org/10.2514/6.2002-3191

[5] Lee, Y., and Baeder, J., "Implicit Hole Cutting—A New Approach to Overset Grid Connectivity," *16th AIAA Computational Fluid Dynamics Conference*, AIAA Paper 2003-4128, 2003. https://doi.org/10.2514/6.2003-4128

[6] Landmann, B., and Montagnac, M., "A Highly Automated Parallel Chimera Method for Overset Grids Based on the Implicit Hole Cutting Technique," *International Journal for Numerical Methods in Fluids*, Vol. 66, No. 6, 2011, pp. 778–804. https://doi.org/10.1002/fld.2292

[7] Rendall, T. C. S., and Allen, C. B., "Parallel Efficient Mesh Motion Using Radial Basis Functions with Application to Multi-Bladed Rotors," *International Journal for Numerical Methods in Engineering*, Vol. 81, No. 1, 2010, pp. 89–105.

[8] Luke, E., Collins, E., and Blades, E., "A Fast Mesh Deformation Method Using Explicit Interpolation," *Journal of Computational Physics*, Vol. 231, No. 2, 2012, pp. 586–601. https://doi.org/10.1016/j.jcp.2011.09.021

[9] Stein, K., Tezduyar, T., and Benney, R., "Mesh Moving Techniques for Fluid-Structure Interactions with Large Displacements," *Journal of Applied Mechanics*, Vol. 70, No. 1, 2003, pp. 58–63.

[10] Stein, K., Tezduyar, T. E., and Benney, R., "Automatic Mesh Update with the Solid-Extension Mesh Moving Technique," *Computer Methods in Applied Mechanics and Engineering*, Vol. 193, Nos. 21–22, 2004, pp. 2019–2032.

[11] Kenway, G. K., Kennedy, G. J., and Martins, J. R. R. A., "A CAD-Free Approach to High-Fidelity Aerostructural Optimization," *Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, AIAA Paper 2010-9231, 2010. https://doi.org/10.2514/6.2010-9231

[12] Liu, X., Qin, N., and Xia, H., "Fast Dynamic Grid Deformation Based on Delaunay Graph Mapping," *Journal of Computational Physics*, Vol. 211, No. 2, 2006, pp. 405–423.

[13] Rendall, T. C., and Allen, C. B., "Unified Fluid–Structure Interpolation and Mesh Motion Using Radial Basis Functions," *International Journal for Numerical Methods in Engineering*, Vol. 74, No. 10, 2008, pp. 1519–1559. https://doi.org/10.1002/nme.2219

[14] Rendall, T. C., and Allen, C. B., "Efficient Mesh Motion Using Radial Basis Functions with Data Reduction Algorithms," *Journal of Computational Physics*, Vol. 228, No. 17, 2009, pp. 6231–6249.

[15] Melville, R., "Dynamic Aeroelastic Simulation of Complex Configurations Using Overset Grid Systems," *Fluids 2000 Conference and Exhibit*, AIAA Paper 2000-2341, 2000. https://doi.org/10.2514/6.2000-2341

[16] De Boer, A., Van der Schoot, M., and Bijl, H., "Mesh Deformation Based on Radial Basis Function Interpolation," *Computers and Structures*, Vol. 85, Nos. 11–14, 2007, pp. 784–795. https://doi.org/10.1016/j.compstruc.2007.01.013

[17] Morris, A., Allen, C., and Rendall, T., "CFD-Based Optimization of Aerofoils Using Radial Basis Functions for Domain Element Parameterization and Mesh Deformation," *International Journal for Numerical Methods in Fluids*, Vol. 58, No. 8, 2008, pp. 827–860. https://doi.org/10.1002/fld.1769

[18] Witteveen, J., and Bijl, H., "Explicit Mesh Deformation Using Inverse Distance Weighting Interpolation," *19th AIAA Computational Fluid Dynamics*, AIAA Paper 2009-3996, 2009. https://doi.org/10.2514/6.2009-3996

[19] Sheng, C., and Allen, C. B., "Efficient Mesh Deformation Using Radial Basis Functions on Unstructured Meshes," *AIAA Journal*, Vol. 51, No. 3, 2013, pp. 707–720. https://doi.org/10.2514/1.J052126

[20] Poirier, V., and Nadarajah, S., "Efficient Reduced-Radial Basis Function-Based Mesh Deformation Within an Adjoint-Based Aerodynamic Optimization Framework," *Journal of Aircraft*, Vol. 53, No. 6, 2016, pp. 1905–1921. https://doi.org/10.2514/1.C033573

[21] Bischof, C. H., Mauer, A., Jones, W. T., and Samareh, J., "Experiences with Automatic Differentiation Applied to a Volume Grid Generation Code," *Journal of Aircraft*, Vol. 35, No. 4, 1998, pp. 569–573. https://doi.org/10.2514/2.2361

[22] Truong, A. H., Oldfield, C. A., and Zingg, D. W., "Mesh Movement for a Discrete-Adjoint Newton-Krylov Algorithm for Aerodynamic Optimization," *AIAA Journal*, Vol. 46, No. 7, 2008, pp. 1695–1704. https://doi.org/10.2514/1.33836

[23] Mader, C. A., Kenway, G. K. W., Yildirim, A., and Martins, J. R. R. A., "ADflow: An Open-Source Computational Fluid Dynamics Solver for Aerodynamic and Multidisciplinary Optimization," *Journal of Aerospace Information Systems*, Vol. 17, No. 9, 2020, pp. 508–527. https://doi.org/10.2514/1.I010796

[24] He, P., Mader, C. A., Martins, J. R. R. A., and Maki, K. J., "DAFoam: An Open-Source Adjoint Framework for Multidisciplinary Design Optimization with OpenFOAM," *AIAA Journal*, Vol. 58, No. 3, 2020, pp. 1304–1319. https://doi.org/10.2514/1.J058853

[25] Secco, N. R., Jasa, J. P., Kenway, G. K. W., and Martins, J. R. R. A., "Component-Based Geometry Manipulation for Aerodynamic Shape Optimization with Overset Meshes," *AIAA Journal*, Vol. 56, No. 9, 2018, pp. 3667–3679. https://doi.org/10.2514/1.J056550

[26] Finlayson, M. A., *ANSYS ICEM CFD User's Manual*, ANSYS, Inc., Canonsburg, PA, Nov. 2013, p. 46.

[27] Steger, J. L., and Rizk, Y. M., "Generation of Three-Dimensional Body-Fitted Coordinates Using Hyperbolic Partial Differential Equations," NASA TM-86753, 1985. https://doi.org/10.1016/0096-3003(92)90073-A

[28] Thompson, J. F., Soni, B. K., and Weatherill, N. P., *Handbook of Grid Generation*, CRC Press, Boca Raton, FL, 1999, Chap. 5.

[29] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., May, D. A., McInnes, L. C., Mills, R. T., Munson, T., Rupp, K., Sanan, P., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H., "PETSc Users Manual," Rev. 3.10, Argonne National Lab. TR ANL-95/11, Lemont, IL, 2018, http://www.mcs.anl.gov/petsc.

[30] Chan, W., Gomez, R., Rogers, S., and Buning, P., "Best Practices in Overset Grid Generation," *32nd AIAA Fluid Dynamics Conference and Exhibit*, AIAA Paper 2002-3191, 2002. https://doi.org/10.2514/6.2002-3191

[31] Chan, W. M., and Buning, P. G., "Surface Grid Generation Methods for Overset Grids," *Computers and Fluids*, Vol. 24, No. 5, 1995, pp. 509–522. https://doi.org/10.1016/0045-7930(95)00003-U

[32] Chan, W. M., "Best Practices on Overset Structured Mesh Generation for the High-Lift CRM Geometry," *55th AIAA Aerospace Sciences Meeting*, AIAA Paper 2017-0362, 2017. https://doi.org/10.2514/6.2017-0362

[33] Chan, W. M., "Strategies Toward Automation of Overset Structured Surface Grid Generation," *23rd AIAA Computational Fluid Dynamics Conference*, AIAA Paper 2017-3451, 2017. https://doi.org/10.2514/6.2017-3451

[34] Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, No. 3, 1988, pp. 233–260. https://doi.org/10.1007/BF01061285

[35] Peter, J. E. V., and Dwight, R. P., "Numerical Sensitivity Analysis for Aerodynamic Optimization: A Survey of Approaches," *Computers and Fluids*, Vol. 39, No. 3, 2010, pp. 373–391. https://doi.org/10.1016/j.compfluid.2009.09.013

[36] Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A., "Scalable Parallel Approach for High-Fidelity Steady-State Aeroelastic Analysis and Adjoint Derivative Computations," *AIAA Journal*, Vol. 52, No. 5, 2014, pp. 935–951. https://doi.org/10.2514/1.J052255

[37] Martins, J. R. R. A., and Hwang, J. T., "Review and Unification of Methods for Computing Derivatives of Multidisciplinary Computational Models," *AIAA Journal*, Vol. 51, No. 11, 2013, pp. 2582–2599. https://doi.org/10.2514/1.J052184

[38] Hascoet, L., and Pascual, V., "The Tapenade Automatic Differentiation Tool: Principles, Model, and Specification," *ACM Transactions on Mathematical Software*, Vol. 39, No. 3, 2013, pp. 20-1–20-43. https://doi.org/10.1145/2450153.2450158

[39] Hascoët, L., and Pascual, V., "TAPENADE 2.1 User's Guide," Institut National de Recherche en Informatique et en Automatique (INRIA) TR 300, Le Chesnay-Rocquencourt, France, 2004, https://hal.inria.fr/inria-00069880/document.

[40] Chan, W., and Gomez, R., III, "Advances in Automatic Overset Grid Generation Around Surface Discontinuities," *14th Computational Fluid Dynamics Conference*, AIAA Paper 1999-3303, 1999. https://doi.org/10.2514/6.1999-3303

[41] Bentley, J. L., "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM*, Vol. 18, No. 9, 1975, pp. 509–517. https://doi.org/10.1145/361002.361007

[42] Kennel, M. B., "KDTREE 2: Fortran 95 and C++ Software to Efficiently Search for Near Neighbors in a Multi-Dimensional Euclidean Space," 2004, https://arxiv.org/abs/physics/0408067.

[43] Kenway, G. K. W., Mader, C. A., He, P., and Martins, J. R. R. A., "Effective Adjoint Approaches for Computational Fluid Dynamics," *Progress in Aerospace Sciences*, Vol. 110, Oct. 2019, Paper 100542. https://doi.org/10.1016/j.paerosci.2019.05.002

[44] Mader, C. A., Martins, J. R. R. A., Alonso, J. J., and van der Weide, E., "ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers," *AIAA Journal*, Vol. 46, No. 4, 2008, pp. 863–873. https://doi.org/10.2514/1.29123

[45] Vassberg, J. C., DeHaan, M. A., Rivers, S. M., and Wahls, R. A., "Development of a Common Research Model for Applied CFD Validation Studies," *26th AIAA Applied Aerodynamics Conference, Guidance, Navigation, and Control and Co-Located Conferences*, AIAA Paper 2008-6919, 2008. https://doi.org/10.2514/6.2008-6919

[46] Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A., "Aerodynamic Shape Optimization Investigations of the Common Research Model Wing Benchmark," *AIAA Journal*, Vol. 53, No. 4, 2015, pp. 968–985. https://doi.org/10.2514/1.J053318

[47] Kenway, G. K. W., and Martins, J. R. R. A., "Multipoint Aerodynamic Shape Optimization Investigations of the Common Research Model Wing," *AIAA Journal*, Vol. 54, No. 1, 2016, pp. 113–128. https://doi.org/10.2514/1.J054154

[48] Brodersen, O., "Drag Prediction of Engine-Airframe Interference Effects Using Unstructured Navier-Stokes Calculations," *Journal of Aircraft*, Vol. 39, No. 6, 2002, pp. 927–935. https://doi.org/10.2514/2.3037

[49] Secco, N. R., and Martins, J. R. R. A., "RANS-Based Aerodynamic Shape Optimization of a Strut-Braced Wing with Overset Meshes," *Journal of Aircraft*, Vol. 56, No. 1, 2019, pp. 217–227. https://doi.org/10.2514/1.C034934

[50] Weller, H. G., Tabor, G., Jasak, H., and Fureby, C., "A Tensorial Approach to Computational Continuum Mechanics Using Object-Oriented Techniques," *Computers in Physics*, Vol. 12, No. 6, 1998, pp. 620–631. https://doi.org/10.1063/1.168744

[51] Jasak, H., Jemcov, A., and Tuković, Z., "OpenFOAM: A C++ Library for Complex Physics Simulations," *International Workshop on Coupled Methods in Numerical Dynamics*, Univ. of Zagreb, Dubrovnik, Croatia, 2007, pp. 47–66.

[52] Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The Complex-Step Derivative Approximation," *ACM Transactions on Mathematical Software*, Vol. 29, No. 3, 2003, pp. 245–262. https://doi.org/10.1145/838250.838251

[53] He, X., Li, J., Mader, C. A., Yildirim, A., and Martins, J. R. R. A., "Robust Aerodynamic Shape Optimization—From a Circle to an Airfoil," *Aerospace Science and Technology*, Vol. 87, April 2019, pp. 48–61. https://doi.org/10.1016/j.ast.2019.01.051

[54] Kennedy, G. J., and Martins, J. R. R. A., "A Parallel Finite-Element Framework for Large-Scale Gradient-Based Design Optimization of High-Performance Structures," *Finite Elements in Analysis and Design*, Vol. 87, Sept. 2014, pp. 56–73. https://doi.org/10.1016/j.finel.2014.04.011

[55] Kenway, G. K. W., and Martins, J. R. R. A., "Multipoint High-Fidelity Aerostructural Optimization of a Transport Aircraft Configuration," *Journal of Aircraft*, Vol. 51, No. 1, 2014, pp. 144–160. https://doi.org/10.2514/1.C032150

[56] Kenway, G. W. K., and Martins, J. R. R. A., "High-Fidelity Aerostructural Optimization Considering Buffet Onset," *Proceedings of the 16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA Paper 2015-2790, 2015.

[57] Brooks, T. R., Kenway, G. K. W., and Martins, J. R. R. A., "Benchmark Aerostructural Models for the Study of Transonic Aircraft Wings," *AIAA Journal*, Vol. 56, No. 7, 2018, pp. 2840–2855. https://doi.org/10.2514/1.J056603

[58] Brooks, T. R., Martins, J. R. R. A., and Kennedy, G. J., "Aerostructural Trade-Offs for Tow-Steered Composite Wings," *Journal of Aircraft*, Vol. 57, No. 5, 2020, pp. 787–799. https://doi.org/10.2514/1.C035699

[59] Gray, J. S., Hwang, J. T., Martins, J. R. R. A., Moore, K. T., and Naylor, B. A., "OpenMDAO: An Open-Source Framework for Multidisciplinary Design, Analysis, and Optimization," *Structural and Multidisciplinary Optimization*, Vol. 59, No. 4, 2019, pp. 1075–1104. https://doi.org/10.1007/s00158-019-02211-z

[60] Gray, J. S., Mader, C. A., Kenway, G. K. W., and Martins, J. R. R. A., "Modeling Boundary Layer Ingestion Using a Coupled Aeropropulsive Analysis," *Journal of Aircraft*, Vol. 55, No. 3, 2018, pp. 1191–1199. https://doi.org/10.2514/1.C034601

[61] Gray, J. S., Kenway, G. K. W., Mader, C. A., and Martins, J. R. R. A., "Aero-Propulsive Design Optimization of a Turboelectric Boundary Layer Ingestion Propulsion System," *2018 AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA Paper 2018-3976, 2018. https://doi.org/10.2514/6.2018-3976

[62] Yildirim, A., Gray, J. S., Mader, C. A., and Martins, J. R. R. A., "Aeropropulsive Design Optimization of a Boundary Layer Ingestion System," *AIAA Aviation Forum*, AIAA Paper 2019-3455, 2019. https://doi.org/10.2514/6.2019-3455

[63] Gray, J. S., Mader, C. A., Kenway, G. K. W., and Martins, J. R. R. A., "Coupled Aeropropulsive Design Optimization of a Three-Dimensional BLI Propulsor Considering Inlet Distortion," *Journal of Aircraft*, Vol. 57, No. 6, 2020, pp. 1014–1025. https://doi.org/10.2514/1.C035845

[64] He, P., Mader, C. A., Martins, J. R. R. A., and Maki, K. J., "An Aerodynamic Design Optimization Framework Using a Discrete Adjoint Approach with OpenFOAM," *Computers and Fluids*, Vol. 168, May 2018, pp. 285–303. https://doi.org/10.1016/j.compfluid.2018.04.012

[65] Li, J., He, S., and Martins, J. R. R. A., "Data-Driven Constraint Approach to Ensure Low-Speed Performance in Transonic Aerodynamic Shape Optimization," *Aerospace Science and Technology*, Vol. 92, Sept. 2019, pp. 536–550. https://doi.org/10.1016/j.ast.2019.06.008

[66] Mangano, M., and Martins, J. R. R. A., "Multipoint Aerodynamic Shape Optimization for Subsonic and Supersonic Regimes," *Journal of Aircraft* (in press). https://doi.org/10.2514/1.C036216

[67] Shi, Y., Mader, C. A., He, S., Halila, G. L. O., and Martins, J. R. R. A., "Natural Laminar-Flow Airfoil Optimization Design Using a Discrete Adjoint Approach," *AIAA Journal*, Vol. 58, No. 11, 2020, pp. 4702–4722. https://doi.org/10.2514/1.J058944

[68] Lyu, Z., and Martins, J. R. R. A., "Aerodynamic Design Optimization Studies of a Blended-Wing-Body Aircraft," *Journal of Aircraft*, Vol. 51, No. 5, 2014, pp. 1604–1617. https://doi.org/10.2514/1.C032491

[69] Chen, S., Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A., "Aerodynamic Shape Optimization of the Common Research Model Wing-Body-Tail Configuration," *Journal of Aircraft*, Vol. 53, No. 1, 2016, pp. 276–293. https://doi.org/10.2514/1.C033328

[70] Kenway, G. K. W., and Martins, J. R. R. A., "Buffet Onset Constraint Formulation for Aerodynamic Shape Optimization," *AIAA Journal*, Vol. 55, No. 6, 2017, pp. 1930–1947. https://doi.org/10.2514/1.J055172

[71] Burdette, D. A., and Martins, J. R. R. A., "Impact of Morphing Trailing Edge on Mission Performance for the Common Research Model," *Journal of Aircraft*, Vol. 56, No. 1, 2019, pp. 369–384. https://doi.org/10.2514/1.C034967

[72] Bons, N. P., He, X., Mader, C. A., and Martins, J. R. R. A., "Multimodality in Aerodynamic Wing Design Optimization," *AIAA Journal*, Vol. 57, No. 3, 2019, pp. 1004–1018. https://doi.org/10.2514/1.J057294

[73] Bons, N. P., and Martins, J. R. R. A., "Aerostructural Design Exploration of a Wing in Transonic Flow," *Aerospace*, Vol. 7, No. 8, 2020, p. 118. https://doi.org/10.3390/aerospace7080118

[74] Dhert, T., Ashuri, T., and Martins, J. R. R. A., "Aerodynamic Shape Optimization of Wind Turbine Blades Using a Reynolds-Averaged Navier–Stokes Model and an Adjoint Method," *Wind Energy*, Vol. 20, No. 5, 2017, pp. 909–926. https://doi.org/10.1002/we.2070

[75] Madsen, M. H. A., Zahle, F., Sørensen, N. N., and Martins, J. R. R. A., "Multipoint High-Fidelity CFD-Based Aerodynamic Shape Optimization of a 10 MW Wind Turbine," *Wind Energy Science*, Vol. 4, No. 2, April 2019, pp. 163–192. https://doi.org/10.5194/wes-4-163-2019

[76] He, P., Martins, J. R. R. A., Mader, C. A., and Maki, K., "Aerothermal Optimization of a Ribbed U-Bend Cooling Channel Using the Adjoint Method," *International Journal of Heat and Mass Transfer*, Vol. 140, Sept. 2019, pp. 152–172. https://doi.org/10.1016/j.ijheatmasstransfer.2019.05.075

S. Fu
Associate Editor