This is a preprint of the following article, which is available from http://mdolab.engin.umich.edu
Ney R. Secco, John P. Jasa, Gaetan K. W. Kenway, and Joaquim R. R. A. Martins. "Component-Based Geometry Manipulation for Aerodynamic Shape Optimization with Overset Meshes", *AIAA Journal*, Vol. 56, No. 9 (2018), pp. 3667-3679. doi:10.2514/1.J056550
The published article may differ from this preprint, and is available by following the DOI above.

Component-based Geometry Manipulation for Aerodynamic Shape Optimization with Overset Meshes

Ney R. Secco, John P. Jasa, Gaetan K. W. Kenway, and Joaquim R. R. A. Martins

Abstract Mesh generation for high-fidelity computational fluid dynamics simulations and aerodynamic shape optimization is a time-consuming task. Complex geometries can be accurately modeled using overset meshes, whereby multiple high-quality structured meshes corresponding to different aircraft components overlap to model the full aircraft configuration. However, from the standpoint of geometry manipulation, most methods operate on the entire geometry rather than on separate components, which diminishes the advantages of overset meshes. To address this issue, we introduce a geometry module that operates on individual components and automatically computes their intersections to update overset meshes during optimization. We apply reverse-mode automatic differentiation to compute partial derivatives across this geometry module, so that it fits into an optimization framework that uses a hybrid adjoint method (ADjoint) to efficiently compute gradients for a large number of design variables. By using these automatically updated meshes and the corresponding derivatives, we optimize the aerodynamic shape of the DLR-F6 geometry while allowing changes in the wing–fuselage intersection. Sixteen design variables control the fuselage shape and 128 design variables determine the wing surface. Under transonic flight conditions, the optimization reduces drag by 15 counts (5%) compared with the baseline design.

Contents

Nomenclature

C_D	drag coefficient
C_L	lift coefficient
D_{fuse}	fuselage diameter
f	function of interest
$\mathbf{g}$	generic vector
n_{fuse}	fuselage shape variables
$\mathbf{R}$	CFD residuals
$\mathbf{r}$	nodal coordinates
t	wing thickness
$\mathbf{W}$	flow-state variables
$\mathbf{X}$	generic inputs or design variables
$\dot{\mathbf{X}}$	seeds of forward automatic differentiation derivative
$\overline{\mathbf{X}}$	seeds of reverse automatic differentiation derivative
z_{wing}	wing shape variables
α	angle of attack
Δx_{wing}	horizontal displacement of the wing
τ_{wing}	wing twist angle
$\boldsymbol{\psi}$	adjoint variables

1 Introduction

With stricter environmental regulations, increased air traffic, and thinning profit margins, airlines and aircraft manufacturers are looking to make the next generation of aircraft as efficient as possible. An airplane configuration may be seen as a group of individual components, such as lifting surfaces, bodies, fairings, and engines; this reflects on how we parametrize the shape of airplanes. Furthermore, from the aerodynamic perspective, junctions between these components cause interference drag due to complex flow features such as separation bubbles. Interference drag depends not only on the overall flow conditions, but also on the detailed shape of the junction.

Low-fidelity tools, such as lifting-line theory and panel codes, can capture general trends associated with individual aircraft components but do not model enough of the physics to accurately resolve the junction flow. Although surrogate models of interference drag [1, 2] can be coupled to low-fidelity tools for preliminary aircraft design [3], they allow only a limited number of design parameters for training, and the predictions are only valid within the training domain of the surrogate model. Therefore, high-fidelity computational fluid dynamics (CFD) simulations, such as Reynolds-averaged Navier–Stokes (RANS) analysis, is essential for detailed analysis and optimization of junction geometries.

Another motivation to use high-fidelity aerodynamic models is that they allow rapid design of aircraft without extensive wind-tunnel testing or other costly evaluation methods, while maintaining an acceptable level of accuracy. Once computational models and analyses are created, several designs can be evaluated at comparatively low cost. These multiple designs require a large number of variables for a detailed parametrization of the aerodynamic shape. Thus, gradient-based optimization techniques are necessary to efficiently search the design space for the optimum configuration. Gradient-based optimization combined with the adjoint method [4] is particularly powerful, since the cost of using the adjoint method to compute gradients does not scale with the number of design variables [5].

CFD requires that the domain be discretized by using meshes, and creating these meshes for complex geometries has historically been challenging, creating a bottleneck in the design process. Multiblock meshes work well for simple geometries but are cumbersome for models with multiple complex features, such as the truss-braced wing. To optimize this type of configuration in previous work, a CFD solver based on structured multiblock meshes was used, but this limits the possible changes in the wing-truss junction geometry due to difficulties in generating and deforming the cells in the concave regions around these junctions [6]. A recurring problem is that, after the mesh-deformation procedures, surface deformations frequently generate

negative-volume cells.

CFD using overset meshes [7] has the potential to overcome this problem because it allows high-quality meshes to be independently generated for each component. These dedicated meshes can also be automatically generated by using, for instance, hyperbolic mesh generation algorithms [8, 9]. Meshes specific to each junction, called *collar meshes*, must also be generated to guarantee that the mesh edges represent the intersection curve after the hole cutting process [10]. In addition to facilitating mesh generation, another advantage of the overset approach is that, because overset meshes are composed of multiple independent structured blocks, we can still use high-quality, memory-efficient structured meshes. Because of this, overset CFD may excel for optimizations where geometry components shift significantly relative to each other, such as when optimizing the location of the wing–fuselage intersection for a conventional airplane, or when independently modifying components during an optimization.

Conversely, these advantages may be hampered if a limited geometry manipulation method is used in the optimization. Although many geometry creation tools exist [11–13], most do not seamlessly integrate geometry description and mesh generation. Furthermore, most of these tools do not calculate derivatives of the geometry description with respect to the design variables, making them inefficient for gradient-based optimization.

A recently proposed solution is to base the reference geometry on a parametrized CAD model and use geometry surrogate models to replace the CAD engine in the optimization framework [14]. Provided the surrogate model is smooth and differentiable, this approach provides analytical derivatives across the geometry manipulation module. However, a large number of training points may be necessary to obtain a valid surrogate for cases with several design variables and complex geometries. To avoid loss of surface information, the user should also be careful when selecting which surface features from the CAD model are exposed to the surrogate model training.

An alternative type of geometry manipulation method used in aircraft design optimization is the free-form deformation (FFD) approach, which parametrizes shape perturbations rather than the shape itself [15, 16]. This approach not only reduces the number of design variables [17] but also allows the use of externally provided geometries in native formats. Nevertheless, this method operates simultaneously on the entire geometry, thereby ignoring valuable information on component subdivision. For instance, in a wing–fuselage configuration, using FFD to translate the wing is impossible without changing the fuselage shape embedded in the same FFD block. Even if the wing and body are embedded in separate FFD blocks, additional nontrivial steps are required to track changes in intersection curves.

The goal of this work is thus to develop a geometry-manipulation approach that allows independent parametrization of shape deformations at the level of both the individual component and the entire aircraft while accounting for changes in intersections between these components. The approach is implemented in a tool that we call *pySurf* that uses unstructured discrete surfaces as input to describe each primary component. These surfaces are used to compute intersections between components and regenerate collar meshes automatically at each optimization iteration. This tool also computes derivatives of mesh points with respect to shape deformation design variables, which enables gradient-based aerodynamic shape optimization. These derivatives are computed by using a hybrid adjoint method that combines analytic methods and reverse-mode automatic differentiation (AD) so the computational cost is largely independent of the number of design variables.

To demonstrate the tool that we have developed, we use a RANS overset solver to optimize the wing–fuselage intersection of a conventional aircraft. We demonstrate the efficacy of the proposed methodology and also provide new insights regarding wing–fuselage junction design for transonic flight conditions.

2 Methodology

As mentioned in the introduction, our aim is to fully exploit the overset mesh capabilities by extending the component-based approach used in overset CFD solvers to the geometry manipulation module within the optimization framework. Consequently, we also need to automatically track changes in junctions among these primary components during optimization to properly modify the collar meshes.

Furthermore, the entire process by which a collar mesh is updated must be differentiated to enable

gradient-based aerodynamic shape optimization. We use reverse-mode AD to efficiently compute the derivatives of the generated mesh points with respect to the design variables, as detailed in Sec. 2.4.

2.1 Overview of collar mesh generation

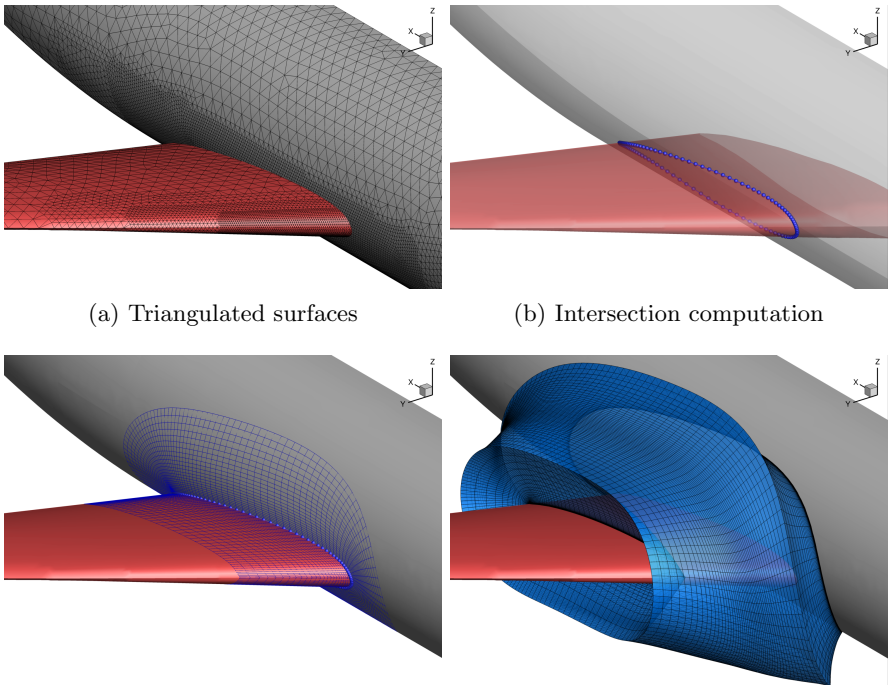
To address the difficulty of automatically creating collar meshes, we developed a new geometry module called pySurf that performs the following sequence of operations:

1. For each primary component, receive triangulated surfaces that represent the full vehicle geometry. pySurf may either read these surfaces from a file at the beginning of the optimization or, at optimization runtime, receive updated surfaces provided by a separate geometry-manipulation module, such as FFDs.

2. Compute the intersections between the triangulated surfaces of the primary components.

3. Automatically generate surface collar meshes for the intersections. These meshes should be structured to be compatible with the CFD solver used in this work.

These operations are represented in steps a–c of Fig. 1. A separate module called pyHyp generates the volume mesh (step d in Fig. 1) and is introduced in Sec. 3.3. The resolution requirements for the triangulated surface are discussed in Sec. 3.



(a) Triangulated surfaces

(b) Intersection computation

(c) Surface mesh marching

(d) Volume mesh extrusion

Figure 1: Steps for generating collar mesh for a wing–fuselage junction. pySurf is responsible for steps a–c. The mesh extrusion module used in step d (pyHyp) is covered in Sec. 3.3.

2.2 Computation of intersection

Once pySurf receives the triangulated surfaces, it must compute the intersections between them. Because of the significant number of triangles used to describe the aircraft geometry, we need a highly efficient intersection algorithm.

We describe here the steps to compute the intersections between two components (for instance, components A and B). This process is repeated for every pair of primary components. First, we compute Cartesian bounding boxes for the two primary components and then determine the intersection between these bounding boxes. Next we flag the triangles from both components that belong to the bounding box intersection region. This step is relatively quick since we just need to compare maximum and minimum values of nodal coordinates against the Cartesian box bounds to flag the triangles.

In the second step, we take the flagged triangles of the component with fewer flagged triangles (for example, component A) and build an alternating digital tree (ADT) [18] to minimize the tree complexity. This tree structure groups discrete elements based on their spatial location so that we can efficiently find the element pairs likely to intersect. We then take each flagged triangle of component B and perform ADT searches to find the flagged triangles from component A that are likely to intersect it.

Finally, in the third step we use fast pairwise triangle-triangle intersection algorithms [19] on the candidate elements given by the ADT searches to identify the two points that determine the intersection line between each pair of triangles. We then concatenate the lines given by the pairwise triangle-triangle intersections to determine the entire intersection curve. These intersection curves then serve as starting curves for generating the hyperbolic surface mesh, as detailed in Sec. 2.3.

An arbitrary number of primary components can be intersected using this method. Each pairwise component intersection may have multiple intersection curves and each is generated and stored for the user to select as possible starting curves for the collar mesh generation.

We also added features dedicated to manipulating these intersection curves, such as merging, splitting, and remeshing, thereby allowing the user to accurately control the topology, the number, and the distribution of nodes describing the intersection. This is important because this curve is the starting point for the hyperbolic surface marching. In addition, to maintain the same number of nodes in the CFD mesh, the number of nodes distributed along the intersection curve remains the same throughout the optimization.

2.3 Generation of hyperbolic surface mesh

Having identified the intersection curves, we can now use established hyperbolic mesh-marching algorithms [8] to produce the surface collar meshes. This type of mesh generation algorithm starts from a baseline curve and then uses a marching scheme to generate the next layer of the surface mesh. This process is repeated until the desired number of layers and mesh extension is reached. One advantage of hyperbolic marching schemes over other mesh generation methods (such as transfinite interpolation [20]) is that they do not require the outer boundary of the mesh domain to be defined, making the process easier to automate, especially for collar meshes.

During mesh marching, we project the mesh points back onto the triangulated surfaces so that the mesh is consistent with the prescribed geometry. For discrete surfaces, finding the nearest surface triangle by brute force would not be tractable, especially when we must regenerate the mesh for each optimization iteration. Thus, we use the ADT algorithm once again to efficiently project the mesh points onto the nearest surface element. The ADT searches provide the surface elements closest to a given point, so that we only need to compare projections on those filtered elements to find the closest one.

We also added features to the standard hyperbolic marching scheme to improve control over mesh generation. For instance, as the surface mesh is marched, the user can choose to preserve special surface features, such as trailing edge corners. First we use line segments to create a discrete representation of these curves. Next, we identify the node from the baseline curve closest to the guide curve segments and then project this node onto the guide curve. In addition, we locally modify the marching equations for this node so that it marches in a direction tangent to the guide curve. This process is repeated for every new layer of the marched mesh.

In addition, we added a feature to preserve the relative node spacings from the baseline curve throughout the entire mesh, which works as follows: After each new layer is generated, we redistribute the nodes of the new layer by using the same relative arc lengths of the nodes in the baseline curve. This redistribution is important when generating meshes near trailing edges because the artificial dissipation associated with the marching scheme tries to smooth the nodal intervals near the trailing edge despite the local high refinement

of the baseline curve.

2.4 Automatic differentiation

We need to efficiently and accurately compute derivatives of the objective and constraints for effective gradient-based aerodynamic shape optimization. Toward this end, we apply the reverse-mode AD throughout the entire analysis chain, which we discuss in detail in Sec. 3.7. Therefore, pySurf also uses the same AD method to internally backpropagate partial derivatives.

In Sec. 2.2, we mentioned that pySurf has several tools to control the intersection curves. During forward execution, pySurf stores all intermediary steps required by the user to generate the appropriate curves for hyperbolic marching, then uses the same steps, but in reverse order, to reverse propagate the derivatives.

The automatically-differentiated code for pySurf is generated by using the Tapenade AD tool [21], but we must still selectively alter the differentiation process of certain pySurf modules to obtain an efficient final code.

For instance, during the forward execution of the surface-projection code used by the hyperbolic surface marching, we store the indices of the triangles that received the projections so that we do not have to repeat the ADT searches during reverse propagation of derivatives. Thus, we only need to differentiate the point-to-triangle projection routine.

A similar procedure can be applied to the intersection computation code. During the forward execution, we use all three steps described in Sec. 2.2. The first two steps use bounding boxes and ADT searches to identify triangles likely to intersect, but they do not compute the intersection curve per se. Therefore, those steps are kept outside of the differentiated routines, because we can store the pairs of intersecting triangles during the forward execution of the code. Thus, only the pairwise triangle-triangle intersection algorithm needs to be differentiated.

We also use an *ad hoc* differentiation approach for the linear system solver inside the hyperbolic marching codes, since we already know the solution for this system from the forward execution. The surface mesh marching module generates the surface mesh layer by layer. Let $\mathbf{r}_i$ be a vector with $3n$ elements representing the Cartesian coordinates of the n nodes of the ith layer of the surface mesh. The nodal coordinates of the next layer are given by

$$\mathbf{r}_{i+1} = \mathbf{r}_i + \Delta\mathbf{r}_i, \tag{1}$$

where the displacement $\Delta\mathbf{r}_i$ is obtained from the solution of a linear system defined by the discretized hyperbolic equations

$$\mathbf{A}_i \cdot \Delta\mathbf{r}_i = \mathbf{b}_i. \tag{2}$$

The linear system of Eq. (2) is solved by using the lower-upper factorization algorithm from LAPACK (Linear Algebra PACKage) [22, 23]. During the reverse execution of the code, we must propagate the derivative seeds from the solution of the linear system ($\Delta\overline{\mathbf{r}}_i$) back to the matrix $\mathbf{A}_i$ and the vector $\mathbf{b}_i$. In a naive AD approach, we would have to build a differentiated version of the linear system solver. However, we can avoid this because we know the solution of the linear system from the forward pass. The corresponding reverse algorithm for Eq. (2) is [24]

$$\mathbf{A}_i^T \cdot \overline{\mathbf{b}}_i = \Delta\overline{\mathbf{r}}_i \tag{3a}$$

$$\overline{\mathbf{A}}_i = -\overline{\mathbf{b}}_i \cdot \Delta\mathbf{r}_i^T. \tag{3b}$$

We first solve the linear system of Eq. (3a) to compute $\overline{\mathbf{b}}_i$ with the same solver used for Eq. (2), then we find $\overline{\mathbf{A}}_i$ by using Eq. (3b). The advantage of this approach is that we only use the native routines from LAPACK without any modification.

We validate the differentiated code in two steps: First, we use the finite-difference test to compare the forward AD results against the original subroutine trends, then we check if the forward AD code and the reverse AD code are consistent with the dot product test [25]. These two tests ensure that the reverse AD code is consistent with the original subroutine. This process is applied to every differentiated module in pySurf. Although we only need the reverse AD version of the code for optimization, these tests also require that we generate the forward AD version with Tapenade.

3 Optimization framework

pySurf on its own is just a geometry module, so it has to be coupled with other analysis tools and an optimizer to create a fully capable framework for aircraft design optimization. Because of the high computational cost of each analysis, high-fidelity aircraft design optimization requires a computationally efficient and massively parallelized multidisciplinary framework. Furthermore, gradient-based optimization is necessary due to the large number of design variables [5, 26, 27]. Thus, this framework must also be able to compute first-order derivatives of both objective and constraint functions with respect to each design variable.

The multidisciplinary design and optimization of aircraft configurations with high fidelity (MACH) framework [28] was developed with these needs in mind. This framework uses Fortran and C++ routines wrapped with a Python interface to perform efficient high-fidelity aerostructural optimization. Specifically for this work, we only use modules relevant to aerodynamic shape optimization, which include a geometry modeler (pyGeo), a volume mesh generator (pyHyp), a volume mesh deformation module (pyWarp), an aerodynamic solver (ADflow), an optimizer [sparse nonlinear optimizer (SNOPT)], and the newly added pySurf module as the collar mesh generator. These modules are described in detail in the following sections.

To use this framework, the user should provide the following for each primary component:

Structured surface mesh with the desired resolution for the CFD analysis. This surface mesh is used to generate the volume mesh with hyperbolic extrusion methods.

Unstructured discrete (triangulated) surface mesh for use as the reference to compute intersections and generate the collar mesh.

FFD block that envelops the surfaces described above and whose control points give the necessary shape control resolution for the optimization problem.

Here we should clarify why we need two descriptions—one structured and the other unstructured—for each primary component. The structured mesh of the primary components has the surface resolution required by the CFD solver, which is already an approximate description of the underlying continuous surface representation. pySurf cannot use the same resolution to march new hyperbolic surface meshes for the collars because this would cause an additional loss of information compared to the original continuous representation. Therefore, the user also needs to provide another surface representation that is finer than that required by the CFD so that the hyperbolic marching algorithm has a description of the surface geometry that is more accurate than the primary structured mesh.

Because this finer triangulated surface mesh is not actually used by the CFD solver, it can be unstructured, making it easier to generate and refine in critical regions, such as high-curvature areas. Because of the high number of Newton searches required to embed all surface points into the FFDs, the use of a heavily refined triangulated surface mainly impacts the setup time of the optimization. The increase in computational time of the subsequent optimization steps is less noticeable because of the linear nature of the FFD method and its derivatives.

The use of triangulated surfaces for automatic mesh generation is recognized as a possible practice for automatic mesh generation [10, 29], because most geometry manipulation packages can output files with this representation. Furthermore, discrete representations are less susceptible to geometry interpretation problems, such as those caused by the use of different geometry kernels when dealing with analytic representations [13].

Further details regarding the framework are also given by Kenway et al. [28], and this framework has been used in several aerodynamic [30, 31] and aerostructural [32] design optimization studies. We now discuss aspects of the framework modules used in this work.

3.1 Geometry modeler—pyGeo

pyGeo is a Python module that generates and manipulates geometries and propagates high-level design variables (twist, chord, shape) to mesh-level changes. This module complements pySurf by handling the primary component geometries, whereas pySurf focuses on intersections and collar meshes. pyGeo uses an FFD approach to parametrize changes in geometry [16]. To set up a new geometry in pyGeo, we first

generate a box of control points surrounding each primary component to define a trivariate B-spline mapping within this volume. Each FFD block can be seen as a structured volume mesh that encompasses the surface that we want to modify. We then use a Newton search to determine the parametric coordinates of the structured surface mesh nodes within their corresponding B-spline boxes. We repeat the same process for the triangulated surface nodes so that both surface representations are parametrized in the same B-spline mapping and thus get consistent displacements.

Any changes applied to the control points can be transferred to the embedded surface nodes using the B-spline mapping. Therefore, the positions of the FFD control points are the primary aerodynamic shape variables of the problem: moving a node from the FFD block deforms the embedded surface.

The FFD deformation process for each component is independent of any other component because we use separate overset meshes, triangulated surfaces, and FFDs. This gives the designer and the optimizer the freedom to control components' variations individually.

The FFD method parametrizes *changes* in geometry rather than the absolute parameters, which avoids the use of CAD-based tools in the optimization process. Furthermore, the B-spline mapping is easily differentiable [17], which allows for efficient computation of the derivatives of the surface node positions with respect to control point positions.

3.2 Collar mesh generator—pySurf

pySurf, which is the module developed in the present work, produces collar meshes between intersected geometry components as described in Sec. 2. pySurf uses the unstructured surface meshes that are updated by pyGeo's FFDs throughout the optimization to compute intersections among primary components.

During each optimization iteration, the surface collar meshes are regenerated by pySurf. Conversely, the surface meshes of the primary component meshes are not recomputed but are directly updated by pyGeo via FFD deformations.

3.3 Volume mesh generator—pyHyp

The previous modules operate on surface meshes; however, we need volume meshes for CFD analysis. We use a hyperbolic volume mesh marching scheme [9] to extrude the surface meshes to produce volume meshes. This mesh generation method is applied to the structured surface meshes of each primary component and collars. This step is only done in the initialization phase to get the baseline volume meshes, which are then deformed by the mesh deformation module (pyWarp) along the optimization since this is less expensive than regenerating volume meshes at each design point.

The same advantages of hyperbolic mesh generation over other methods previously discussed for surface meshes apply here, especially those regarding the degree of automation and robustness. The main user-defined parameters are the height of the initial layer of cells and the extrusion distance. The selected marching distance should allow a reasonable amount of overlap between the collar mesh and the primary component meshes. Additionally, cells within the collar mesh should be smaller than the cells in the primary component meshes to preserve collar mesh cells during the hole cutting process. We usually achieve this by using an initial cell height for the collar meshes equal to 95% of the cell height of the primary components. The surface meshes are extruded to a small extension (such as three mean aerodynamic chords). One example of hyperbolic mesh extrusion was already shown in Fig. 1.

After the extrusion of all surface meshes, pyHyp uses the bounding box of these near-field volume meshes to create a background mesh that reaches an appropriate distance for the far-field boundary conditions. The background mesh generation process is discussed in more detail in Sec. 4.2.

pyHyp is not differentiated because it is used only in the initialization step and thus is not part of the optimization cycle itself, as discussed in Sec. 3.7.

3.4 Volume mesh deformation—pyWarp

Within the optimization loop, we use pyWarp to deform the initial volume meshes generated by pyHyp based on the updated component surfaces. pyGeo provides surface node updates for the primary components based on the FFD deformations, whereas pySurf provides updated surface nodes for the collar meshes based on

the recomputed intersections. Then pyWarp propagates the deformations of the surface nodes to the volume nodes using an inexact explicit interpolation algorithm to reduce computational cost [33].

Given a baseline surface mesh and its corresponding deformed configuration, we can compute displacements and surface normal rotations for each surface node. Each surface node defines a displacement field in the volume based on its own displacements and rotations. The displacement of each volume node is computed as a weighted average of the displacements predicted by the displacement fields of the surface nodes at the volume node location. The weights of the averaging process decay with the distance between the volume node and the corresponding surface node.

This mesh deformation procedure is also applied in a componentwise manner, so that each overset mesh is deformed based only on its own wall surfaces. For instance, the wing surface nodes only affect the deformation of the wing volume mesh nodes, the fuselage surface nodes only deform the fuselage volume mesh nodes, and the collar surface nodes only deform the collar volume mesh nodes. This allows for relative motion between primary components. In other words, if the wing is translated with respect to the fuselage, the movement of the wing volume mesh nodes is not influenced by the fixed position of the fuselage surface nodes. The background mesh remains fixed throughout the optimization because it has no associated surface.

The component-based deformation methodology applied to overset meshes can be advantageous in geometries with oblique junction angles between two distinct components. Volume cells in the gap between these components do not have opposing surfaces driving their displacements, thereby allowing for additional freedom of movement. Furthermore, the overset mesh cells in the junction regions are of higher quality than a patched multiblock mesh, which allows the former to tolerate a wider range of deformations before the mesh becomes invalid for CFD analyses.

Negative volume cells may still arise in the collar mesh if the junction corner angle becomes too severe during the optimization. Alternative approaches such as regenerating the mesh or changing the mesh topology could avoid this issue. However, we do not explore these options in this work because acute junction angles generally increase interference drag [1] and thus may not be favored by the aerodynamic shape optimization framework.

pyWarp is also automatically differentiated by using Tapenade in reverse AD mode so it can compute the derivatives of volume node positions with respect to surface node movements.

3.5 CFD solver—ADflow

Once volume meshes are generated and warped to reflect the changes in geometry, we use a CFD solver to perform aerodynamic analysis on the new configuration and to obtain lift, drag, and moment coefficients. These coefficients are then used by the objective and constraint functions in the optimization problem.

The aerodynamic solver used in this work is a newly implemented version of SUmb called ADflow [34, 35]. It is capable of solving Euler, laminar Navier–Stokes, and RANS equations in multiblock structured overset meshes in a parallelized fashion using a second-order cell-centered finite volume formulation. The inviscid fluxes are discretized with artificially dissipated central-differencing, whereas the viscous fluxes use standard central-differencing.

ADflow initially solves steady problems by using time-marching schemes (such as the diagonalized diagonal dominant alternating direction implicit scheme [36] or the multistage explicit Runge–Kutta scheme [37]) to approach the basin of attraction. It then switches to a Newton–Krylov algorithm to converge to the steady solution [38]. The switching criterion is defined as the point when the time residuals drop below a user-specified threshold.

This CFD solver uses implicit hole cutting [39, 40] to determine which cells should be blanked, interpolated, or actually computed. The interpolated cell values are computed using a first-order interpolation method. Once the flow converges, zipper meshes are used to fill the surface gaps, yielding a watertight surface for the integration of the aerodynamic forces and moments [41]. In this work, we update the overset connectivities and zipper meshes at each optimization iteration. Further details regarding the overset implementation in ADflow are presented by Kenway et al. [42].

ADflow's drag predictions were previously evaluated in the Sixth Drag Prediction Workshop (DPW6) [43]. One important conclusion from DPW6 was that the standard Spalart–Allmaras (SA) turbulence model [44]

could overestimate the size of separation bubbles in wing–fuselage junctions [45], and that this could be mitigated by applying modifications to the turbulence model, such as the rotation correction [46], and the quadratic constitutive relation (QCR) [47]. Therefore, we use the SA-R-QCR2000 turbulence model for all simulations in this work.

A discrete adjoint approach gives the derivatives of the aerodynamic forces and moments with respect to the nodal coordinates [35, 48]. We use Tapenade's reverse-mode AD to compute the partial derivatives of the CFD problem, as explained in Sec. 3.7. The derivatives currently computed by ADflow assume frozen overset interpolation weights. In other words, the linearized code does not consider how changes in mesh coordinates affect the overset interpolation weights. This discrepancy becomes less significant as the overset mesh is refined because flow state variations within the interpolation stencils become smaller. Nevertheless, the interpolation weights are updated in the next forward execution of the code.

3.6 Optimizer—SNOPT

We use SNOPT (Sparse Nonlinear Optimizer) [49], which is a gradient-based optimizer that implements the sequential quadratic programming method. The user must provide functions of interest and their gradients. SNOPT can handle large-scale nonlinear optimization problems with thousands of constraints and design variables, making it suitable for aerodynamic shape and aerostructural optimizations [28, 31, 32, 42].

3.7 Derivative computation throughout the framework

Aerodynamic shape optimization usually involves a number of design variables that is much greater than the number of functions of interest. For this scenario, the adjoint method is an efficient way to compute derivatives [5, 35, 50]. However, a significant effort is required to implement this method because it needs partial derivatives of the residual equations and other quantities used in the CFD code.

Let $f = f(\mathbf{X}, \mathbf{W})$ be a function of interest for the optimization problem (such as drag coefficient). This function depends on the design variables $\mathbf{X}$ and the flow state variables $\mathbf{W}$. The flow state variables are obtained from the solution of the discretized flow equations: $\mathbf{R}(\mathbf{X}, \mathbf{W}) = \mathbf{0}$. The total derivative of the function f with respect to the design variables is

$$\frac{\mathrm{d}f}{\mathrm{d}\mathbf{X}} = \frac{\partial f}{\partial \mathbf{X}} - \boldsymbol{\psi}^T \cdot \frac{\partial \mathbf{R}}{\partial \mathbf{X}}, \tag{4}$$

where the adjoint variables ψ can be obtained by solving the discrete adjoint system:

$$\left[\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\right]^T \cdot \boldsymbol{\psi} = \left[\frac{\partial f}{\partial \mathbf{W}}\right]^T. \tag{5}$$

Our optimization framework uses a hybrid-adjoint approach [35], where the partial derivatives in Eqs. (4) and (5) are computed using the reverse-mode AD of Tapenade.

Figure 2 shows the relationships between the reverse AD modules of the analysis chain. The backpropagation of derivatives starts at the last code of the analysis chain, which is the CFD solver. For given derivative seeds of the CFD residuals ($\overline{\mathbf{R}}$) and function of interest ($\overline{f}$), ADflow uses reverse AD versions of its subroutines to obtain derivative seeds for three parameter sets: flow state variables ($\overline{\mathbf{W}}$), volume mesh nodes ($\overline{\mathbf{r}}_{\mathrm{vol}}$), and flow condition design variables ($\overline{\mathbf{X}}_{\mathrm{aero}}$), such as angle of attack. Lyu et al. [48] give specific details on differentiating the RANS equations in ADflow.

The volume mesh derivative seeds ($\overline{\mathbf{r}}_{\mathrm{vol}}$) are split based on their corresponding components (such as wing, fuselage, and collar), and then transformed into surface node derivatives ($\overline{\mathbf{r}}_{\mathrm{surf}}$) by pyWarp. Next, pySurf receives the derivatives of the collar surface mesh points and executes its own reverse-mode AD to backpropagate the derivative information to the triangulated surface nodes ($\overline{\mathbf{r}}_{\mathrm{tria}}$). Then pyGeo accumulates the derivatives coming from the triangulated surfaces (given by pySurf) and from the structured surface meshes (given by pyWarp) of each primary component. Finally, pyGeo uses the FFD mapping to backpropagate derivatives to the FFD control points, which are translated into derivatives with respect to design variables ($\overline{\mathbf{X}}_{\mathrm{w}}$ for wing design variables and $\overline{\mathbf{X}}_{\mathrm{f}}$ for fuselage design variables).

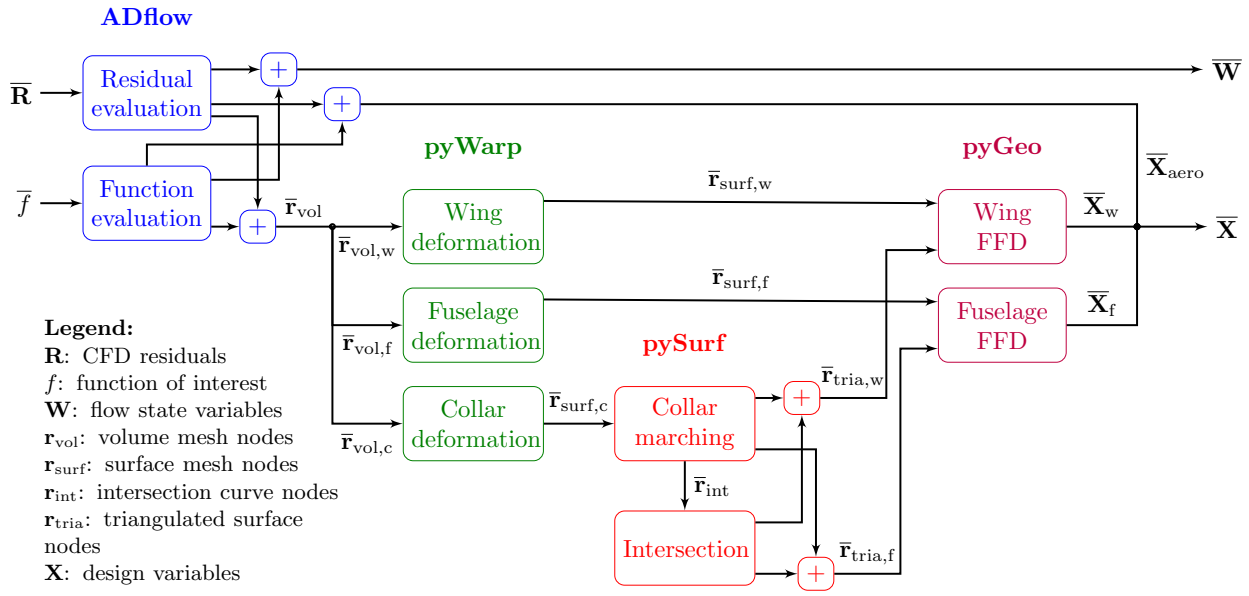


Figure 2: Backpropagation of reverse AD derivative seeds throughout the analysis chain. The plus signs indicate that the derivative seed should be accumulated from multiple sources. Subscripts 'w', 'f', 'c', and 'aero' indicate that quantities refer to wing, fuselage, collar, and flow conditions, respectively.

The derivative seeds of the geometric design variables of the wing and fuselage, ($\overline{\mathbf{X}}_\mathrm{w}$ and $\overline{\mathbf{X}}_\mathrm{f}$), are then concatenated with the flow condition design variables ($\overline{\mathbf{X}}_\mathrm{aero}$) to yield the derivative seeds of the complete design variable vector ($\overline{\mathbf{X}}$), closing the reverse AD propagation. pyHyp is not included in Fig. 2 because it is only used during the initialization step, not during the optimization itself.

We now explain how to use the chain of Fig. 2 to solve Eqs. (4) and (5). The reverse derivative seeds from Fig. 2 are related as follows:

$$\overline{\mathbf{W}} = \left[\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\right]^T \cdot \overline{\mathbf{R}} + \left[\frac{\partial f}{\partial \mathbf{W}}\right]^T \cdot \overline{f}, \tag{6a}$$

$$\overline{\mathbf{X}} = \left[\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right]^T \cdot \overline{\mathbf{R}} + \left[\frac{\partial f}{\partial \mathbf{X}}\right]^T \cdot \overline{f}. \tag{6b}$$

If we feed the reverse AD chain with $\overline{\mathbf{R}} = \mathbf{0}$ and $\overline{f} = 1$, then the resulting $\overline{\mathbf{W}}$ will be the right-hand side of Eq. (5). Now, if we use $\overline{\mathbf{R}} = \mathbf{g}$ and $\overline{f} = 0$, then $\overline{\mathbf{W}}$ will be the matrix-vector product of the left-hand side of Eq. (5) for an arbitrary vector $\mathbf{g}$. Therefore, we can use matrix-free linear system solvers to solve Eq. (5) and obtain $\boldsymbol{\psi}$. In this work we use the generalized minimal residual method (GMRES) [51] implementation in PETSc [52–54] to solve the linear system. Finally, if we apply $\overline{\mathbf{R}} = -\boldsymbol{\psi}$ and $\overline{f} = 1$ to the reverse AD chain, we get, according to Eqs. (4) and (6b), $\overline{\mathbf{X}} = [\mathrm{d}f/\mathrm{d}\mathbf{X}]^T$, which is the total derivative of the function of interest required by the optimizer.

This procedure must be applied for every function of interest to assemble and solve the corresponding adjoint systems. Because we use reverse AD throughout the entire analysis chain, the cost of computing derivatives scales with the number of functions of interest and not with the number of design variables.

3.8 Noise in functions of interest

Preliminary tests of the optimization framework showed that the functions of interest obtained from the CFD evaluations contain noise. However, in the code verification steps, no significant noise appeared in the

variations in mesh point locations after mesh regeneration using the triangulated surfaces. Therefore, we suspect that the noise originates from changes in the overset connectivity.

To clarify this point, we generate a simple geometry consisting of a cylindrical fuselage and a tapered, swept wing based on that of a Boeing 717, which is then simulated in transonic flow conditions (Mach 0.8). We define a single design variable for the geometry: the longitudinal wing position with respect to the fuselage (Δx_{wing} from Fig. 3). The fuselage mesh remains fixed while the wing mesh is horizontally translated according to the value of the design variable.

The intersection line retains the same shape because the wing is translated in the longitudinal direction along the cylindrical fuselage. Therefore, there is no need to regenerate the collar mesh using the triangulated surfaces, though the effects of changes in overset connectivity remain. To amplify the effects of changes in connectivity, the overset mesh used for the analysis has only 0.3 million cells.

We next run flow simulations for a range of wing positions; the results are shown in Fig. 4. The function shows noisy patterns with amplitudes on the order of tenths of drag counts for small wing displacements (see right side of Fig. 4). The noise drops to the order of hundredths of drag counts if the mesh is refined to 1.3 million cells because the interpolation layers become thinner. Therefore, we conclude that the main contribution to the noise comes from changes in overset connectivity.

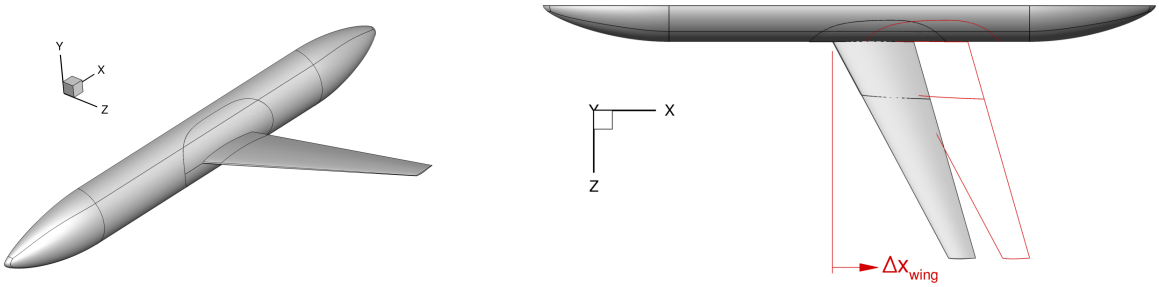


Figure 3: Transonic configuration used to study wing translation. The design variable Δx_{wing} controls the horizontal displacement of the wing.

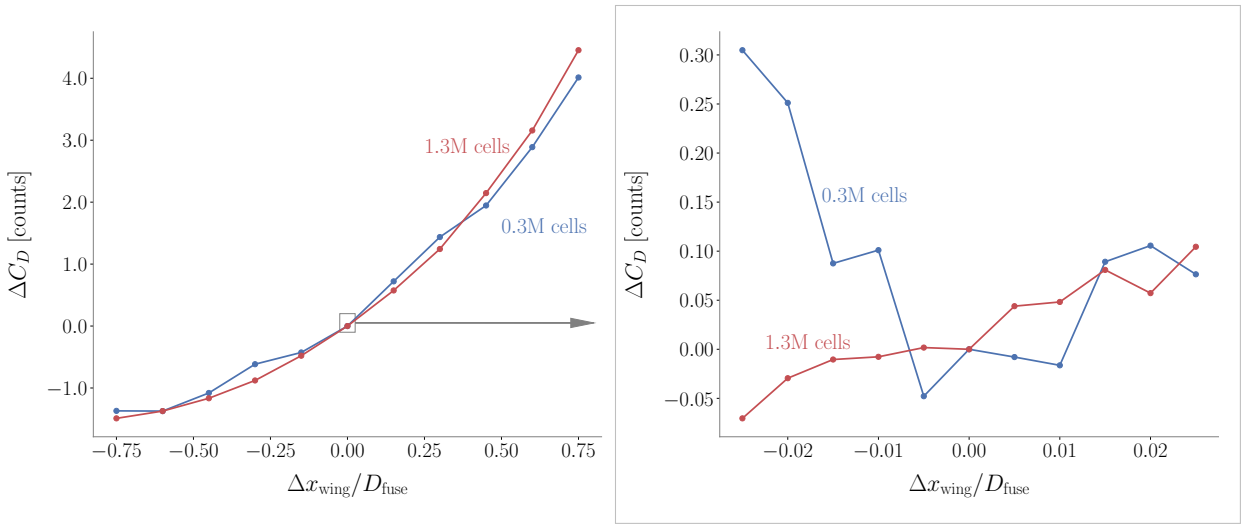


Figure 4: Variations in drag for different horizontal wing positions normalized by the fuselage diameter (D_{fuse}). The right plot shows drag variations for small displacements. The noise caused by changes in overset connectivity decreases as the mesh is refined.

4 DLR-F6 optimization

To demonstrate the capabilities of pySurf, we use it to optimize the aerodynamic shape of the DLR-F6 wing–fuselage configuration [55]. This model was used extensively to generate benchmarks for drag prediction [56, 57] and as a baseline geometry for aerodynamic shape optimization [58–60].

In the baseline DLR-F6 configuration, a separation bubble appears at the trailing edge of the wing–fuselage junction at $C_L = 0.5$ [56], which makes this problem particularly interesting for optimizing the wing–fuselage junction. A fairing was designed for the Third Drag Prediction Workshop (DPW3) to remove the recirculation region and improve drag convergence studies [61]; this configuration is called DLR-F6-FX2B. The separation bubble provides motivation to use the MACH framework, now with the pySurf module, to optimize the aerodynamic shape of the DLR-F6 wing–fuselage junction.

Since pySurf enables independent manipulation of the wing and fuselage, we define three major groups of geometric design variables: wing twist, wing shape, and fairing shape variables. We also use angle of attack as a design variable so that the optimizer has an additional degree of freedom to match the desired lift coefficient. The objective of the optimization is to minimize the drag coefficient (C_D) at $C_L = 0.5$, $M = 0.75$, and $Re = 5 \times 10^6$, following the same standards as in DPW3.

In addition to the lift constraint, we also consider thickness constraints for the wing and a *bubble-gum* constraint for the fuselage, which means that the fuselage shape cannot go inside its original outer-mold line. Table 1 summarizes the overall optimization problem.

To better understand the behavior of the fairing design variables, we propose the two optimization problems listed in Table 2. In the first problem, only the fairing design variables are active (Problem F). Conversely, the second problem (Problem F + T + S) uses all design variables and constraints listed in Table 1 to allow simultaneous optimization of the wing and the junction.

Table 1: DLR-F6 aerodynamic shape optimization problem.

	Variable/function	Description	Quantity
Minimize	C_D	Drag coefficient	
with respect to	α	Angle of attack	1
	n_{fuse}	Normal displacement of fuselage FFD control points	16
	τ_{wing}	Wing section twist	8
	z_{wing}	Vertical displacement of wing FFD control points	128
		Total design variables	**153**
subject to	$C_L = 0.5$	Lift constraint	1
	$t/t_{\text{init}} \geq 0.99$	Wing thickness constraint	100
	$\Delta z_{\text{wing,TE,upper}} = -\Delta z_{\text{wing,TE,lower}}$	Fixed trailing edge	8
	$\Delta z_{\text{wing,LE,upper}} = -\Delta z_{\text{wing,LE,lower}}$	Fixed leading edge	8
	$0.00 \text{ m} \leq n_{\text{fuse}} \leq 0.05 \text{ m}$	Variable bounds	16
		Total constraints	**133**

Table 2: Identification tags for the baseline case and each optimization problem.

Tag	Active design variables	Description
B	α	C_L matching for the baseline configuration
F	α, n_{fuse}	Fairing optimization
F + T + S	α, n_{fuse}, τ_{wing}, z_{wing}	Fairing, wing twist, and wing shape optimization

4.1 Geometric design variables and constraints

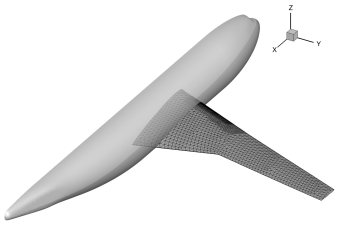
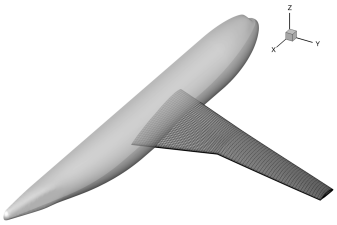
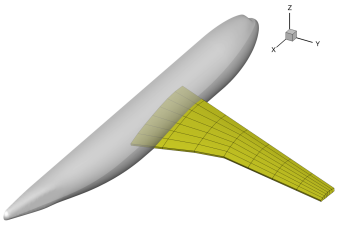
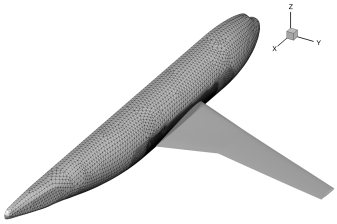
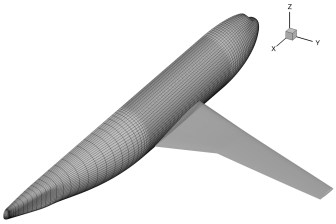
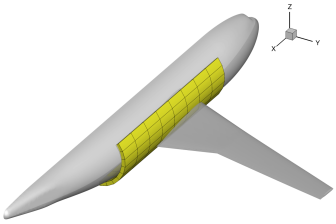
The geometric design variables affect the position of the FFD control points. The wing FFD is composed of a volumetric block with 8 spanwise by 8 chordwise by 2 vertical nodes (see Table 3). The 8 wing twist variables (τ_{wing}) apply uniform rotation around the quarter-chord to each chordwise section of the FFD block. The

128 wing shape variables (z_{wing}) move each control point of the FFD in the vertical direction to control the airfoil shape. To avoid shear twist, the control points on either side of the upper and trailing edges are constrained to move equal amounts but in opposite directions. This ensures that the airfoil chordline is affected only by twist variables and not by shape variables [62].

Because this is a purely aerodynamic shape optimization problem, we also need to enforce thickness constraints to obtain a realistic design. In this case we define a grid of 10×10 points over the wing planform and then require the thickness at these points to remain at or above 99% of the baseline thicknesses.

The fuselage FFD consists of a block of $8 \times 6 \times 2$ nodes that encompasses the region around the wing intersection (see Table 3). The nodes in the outer two layers of the FFD are fixed to guarantee C_1 continuity with the undeformed part of the fuselage [15], what leaves a total of 16 free control points. These control points are only allowed to move in a direction normal to the fuselage surface and can only move away from the center of the fuselage, thereby ensuring the bubble-gum constraint.

Table 3: Inputs required by pySurf; these are generated by ICEM CFD using the original IGES representation of the DLR-F6 model.

	Triangulated surfaces	Structured surface meshes	FFD blocks
Wing			
Fuselage			

4.2 Problem setup

To set up the analysis and optimization cycle in the MACH framework, we must provide triangulated surfaces, structured surface meshes, and FFD boxes for all primary components. We therefore import the IGES files of the DLR-F6 configuration in ICEM CFD [63] to generate these inputs, which are listed in Table 3.

The wing structured surface mesh has 127,488 quadrilaterals, and the fuselage structured surface mesh has 105,344 quadrilaterals. The wing and fuselage triangulated surfaces are refined in the high-curvature regions (such as the leading edge), where we expect the collar mesh to grow. In this optimization, the wing triangulated surface has 9,948 elements, whereas the fuselage triangulated mesh has 19,902 elements, making a total of 29,850 triangles, which are mainly concentrated in the region of the wing–fuselage intersection.

Refining the triangulated surface to 1.3 million triangles for the collar mesh generation causes an average displacement of 0.02 mm for the CFD mesh nodes at the wing–fuselage intersection line and a variation of 0.08 drag counts for the aerodynamic analysis of the baseline configuration, as shown in Fig. 5. Therefore, the optimization results obtained with the former triangulated surface fall within analysis and manufacturing tolerances.

While generating the triangulated surface, we also generate line segments to outline important features of the geometry, such as the two corners of the blunt trailing edge. We project the marched collar mesh

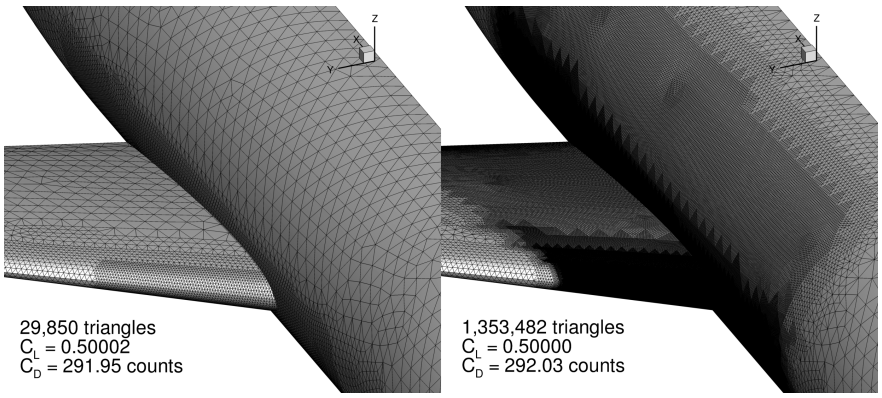


Figure 5: Effect of refinement of triangulated surface over CFD results. The coarse triangularization is used for optimization.

nodes to the line segments to ensure that the corresponding features are represented in the CFD mesh, as explained in Sec. 2.3.

In the next step, we use pySurf to perform the following operations on the triangulated surfaces to generate the collar mesh:

1. Compute the intersection between the wing and fuselage triangulated surfaces.
2. Split the intersection curve at the wing leading edge and at the trailing edge corners; this splits the intersection curve into three segments: upper skin, lower skin, and blunt trailing edge.
3. For each of these segments, create node distributions that are appropriate for CFD analysis (for instance, with refined leading and trailing edges).
4. Merge the three remeshed curve segments back into a single curve.
5. Use the merged curve as a starting feature for the hyperbolic surface mesh marching on the wing and on the fuselage to generate the collar mesh.

pySurf saves the order of these operations for the reverse propagation of derivatives. The execution of these steps generates the collar mesh shown in Fig. 1.

In the initialization step, we use pyHyp to automatically extrude the surface meshes into near-field volume meshes. The extrusion parameters, such as initial cell size and growth ratios, follow the DPW3 guidelines for the coarse mesh level. The near-field meshes are extruded up to 2.5 times the mean aerodynamic chord of the model.

The background mesh, which extends the domain to the far-field, is generated in two steps: First, we generate a block of Cartesian cells to fill the bounding box defined by the near-field meshes. Next, we use hyperbolic extrusion once again to march the outer surface of this Cartesian mesh to reach 100 mean aerodynamic chords, which is the far-field distance specified by DPW3. The final overset mesh has 1.1 million cells.

4.3 Optimization results

We first optimize the DLR-F6 configuration only with respect to the fuselage shape variables (Problem F in Table 2). All other design variables presented in Table 1 are fixed in this problem. The center of Figure 6 shows the optimized fairing for Problem F.

The optimized configuration reduces the drag by approximately five counts. The mesh deformation method discussed in Sec. 3.4 generates negative volume cells in a single design point evaluation during the

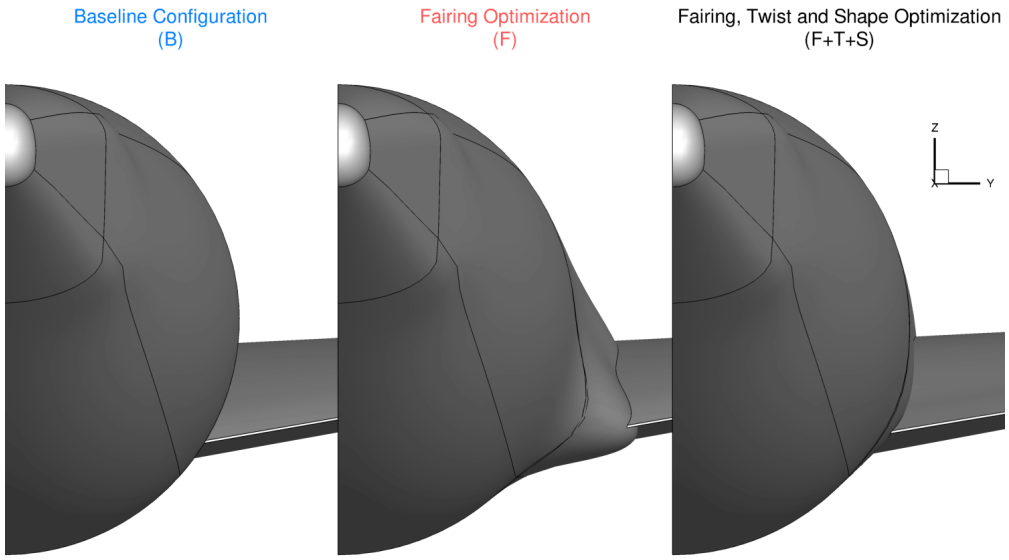


Figure 6: Rear views comparing fairing sizes obtained from different optimization problems. The fairing shrinks as the optimizer gains more control over the wing properties.

optimization, when the volumes of three cells become negative. Nevertheless, the optimizer adjusts the size of the line search step and proceeds without further negative volume issues.

Some fairing design variables reach the upper bounds for n_{fuse} specified in Table 1. The fairing extends even more if we relax these bounds, but excessive stretching leads to problems involving mesh deformation and overset hole cutting. This behavior is explained by noting that, since the optimizer does not have the freedom to modify the wing geometry, it tries to use the fairing to change the wing's lift distribution. The fairing is enlarged to unload the inboard region of the wing and shift the lift distribution toward the wing tip. As shown in Fig. 7, the increased outboard load is closer to an elliptical distribution, which decreases the lift-induced drag.

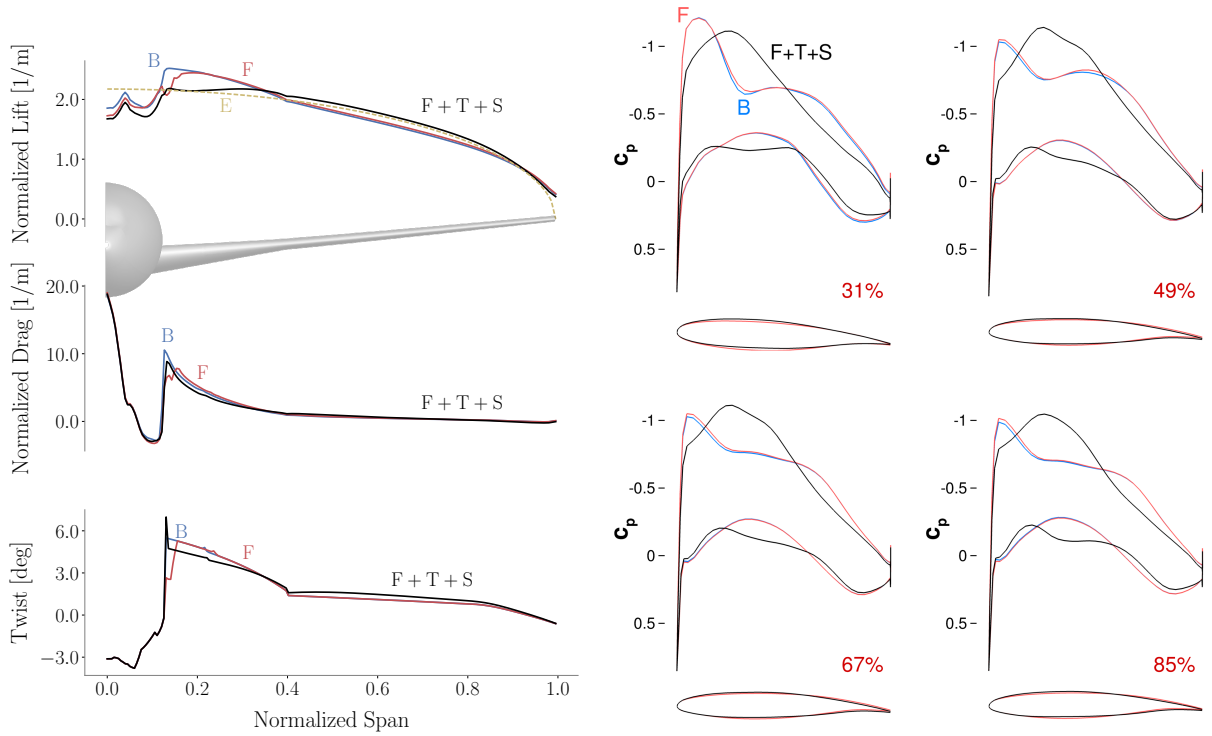
Next, we solve Problem F + T + S listed in Table 2 to investigate the effect of adding the wing design variables. No negative volume cells develop in this second optimization case. Table 4 lists various relevant performance figures regarding the two optimizations, such as optimality and wall time. Neither optimization attains the desired reduction in optimality because both terminate due to numerical difficulties caused by noise in the functions of interest, as explained in Sec. 3.8. The noise becomes a problem when the optimizer is fine tuning the design near the optimum, where variations indicated by the gradients become smaller than the noise amplitude. Nevertheless, for each problem, the optimizer still reduces the drag compared with the baseline configuration.

Table 4 shows that increasing the number of design variables tends to reduce the drag. The full optimization (F + T + S) reduces drag by 15 counts (i.e., 5%) compared with the baseline configuration.

Table 4: Performance figures for the baseline configuration and for each optimization problem.

Problem	C_L	C_D (counts)	C_D reduction (%)	Initial optimality	Final optimality	Wall time (hours)	Processors
B	0.5002	291.99	0.00	-	-	0.2	64
F	0.4993	286.76	1.79	7.4×10^{-1}	1.4×10^{-1}	1.2	64
F + T + S	0.5000	276.85	5.19	1.2×10^{-2}	4.7×10^{-3}	6.0	128

Figure 6 shows how the optimized fairing geometry changes upon the activation of wing twist and wing shape design variables. Including more degrees of freedom in the optimization reduces the fairing size. Since the optimizer has additional variables to tailor the wing, the fairing variables are used only for local



(a) Lift, drag, and twist distributions

(b) Pressure coefficient distributions at span stations

Figure 7: Comparison of aerodynamic characteristics of optimized configurations. (B) is the baseline configuration and (F) is the fairing optimization. The inclusion of twist variables (T) allows the optimizer to achieve more efficient lift distributions that are closer to an elliptical distribution, while adding wing shape variables (S) results in smoother pressure distributions. The equivalent elliptical distribution (E) is also shown.

adjustments to the wing–fuselage junction flow pattern, which requires smaller deformations. Therefore, the fairing design variables no longer reach the prescribed bounds.

Figure 7 shows the lift and drag distributions for the different optimized configurations. The inclusion of twist variables allows the optimizer to reach a more elliptical lift distribution by reducing the wing mounting angle while increasing the incidence at the outboard sections. The introduction of wing shape design variables removes shocks and results in designs with smoother lift distributions (see Fig. 7b), because the optimizer can fine tune the thickness and camber along the wing.

Figure 8 shows the normal displacements on the fuselage surface for the two optimization problems. To prevent recirculated flow in both problems, the optimizer adds two bumps on the fuselage: one near the leading edge and another near the trailing edge.

The fairing design variables effectively remove the separated region of the junction trailing edge in all optimization problems in which they are active, as seen in Fig. 9.

Finally, we show that the drag reduction of the optimized configurations is still present at finer mesh levels. Figure 10 plots the mesh convergence study for the baseline and optimized configurations. The relative ranking of the configurations in terms of drag remains the same for all mesh levels.

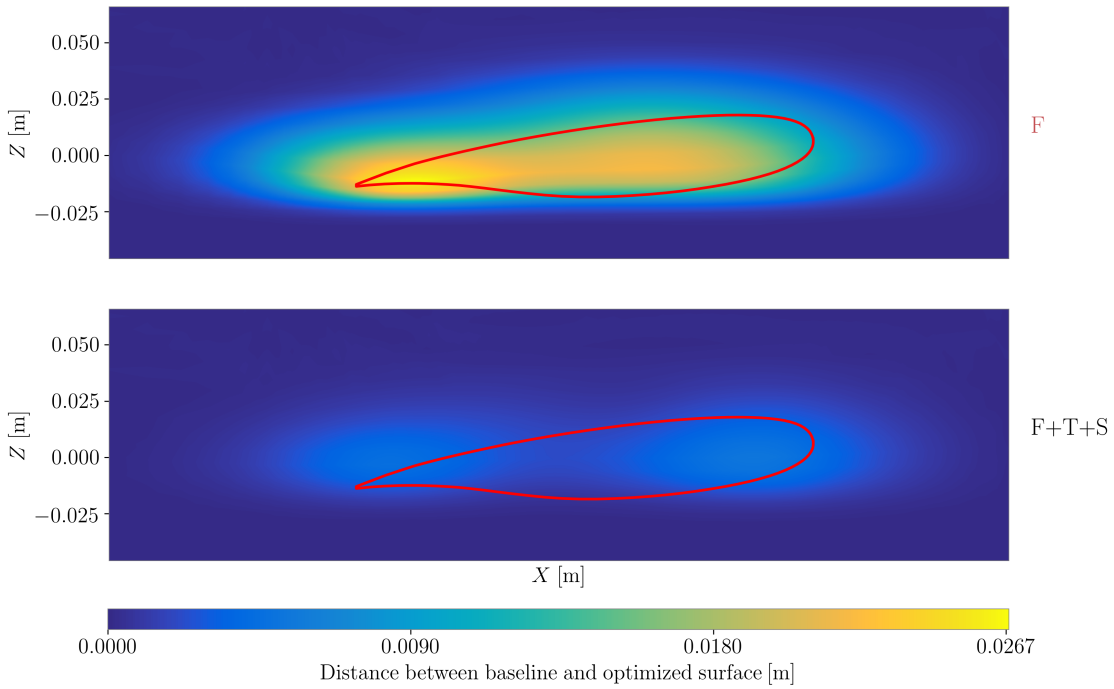


Figure 8: Contour plot showing the distance over which the fuselage surface moves during optimization. The red line represents the wing–fuselage intersection curve of the baseline configuration. To reduce reversed-flow regions, the optimizer extends the fairing both at the leading and trailing edges. When the optimizer controls only the fairing, the fuselage surface shifts a relatively large distance.

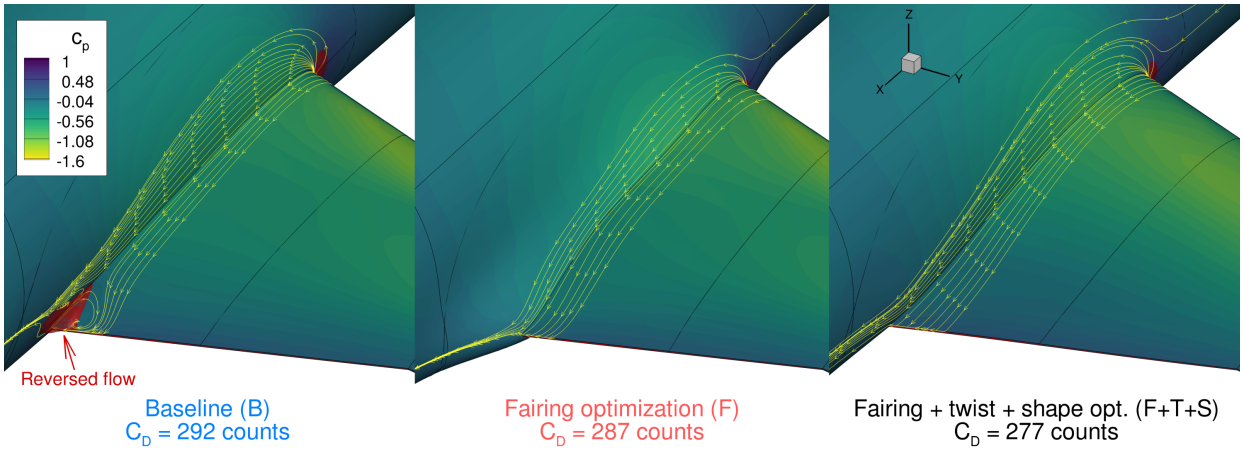


Figure 9: Trailing edge of wing–fuselage junction after each optimization problem. Red regions indicate reversed flow. The redesigned fairings eliminate the recirculation bubble in both optimizations.

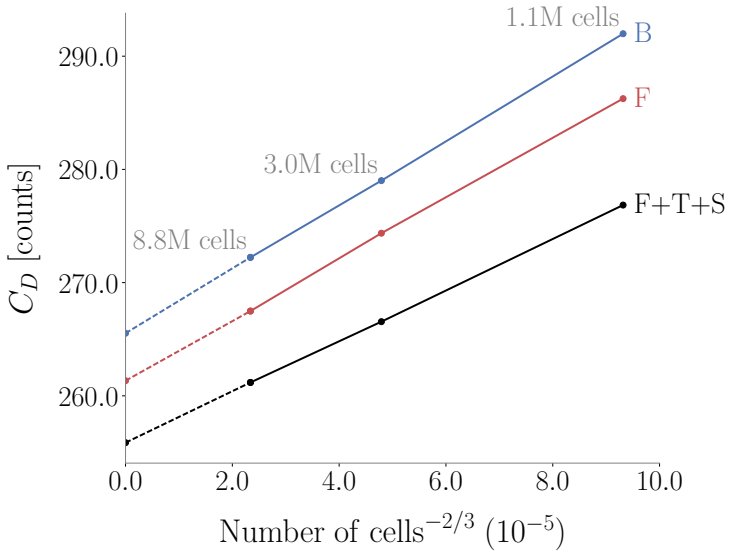


Figure 10: Mesh refinement study for baseline (B) and optimized configurations (F and F + T + S). Dashed lines represent continuum estimates. The optimized design improvements are still present at finer mesh levels.

5 Conclusions

A new geometry module, pySurf, was implemented to automate collar mesh generation based on intersections and also to independently manipulate geometry components in problems involving aerodynamic shape optimization. The intersection computation and surface mesh generation were differentiated to enable the use of gradient-based optimization techniques for a rapid design cycle.

It was demonstrated that independent component manipulation, mesh deformation, and automatic collar mesh generation were feasible for optimization tasks. It was also found that changes in the overset connectivities cause numerical noise in the functions of interest. This noise level can be reduced by refining the overset meshes. It was verified that the optimizer still manages to improve the design, even though it does not strictly meet the optimality criteria.

The new geometry manipulation approach was used to design the wing–fuselage junction of the DLR-F6 and it was shown that a fairing-alone optimization might try to solve issues from the overall configuration, which is unproductive. Adding wing design variables provides more options for the optimizer to solve these issues, leaving the task of locally adjusting the wing–fuselage junction flow to the fairing design variables, as originally intended. Therefore, it was concluded that one should include as many variables as possible to avoid this issue. When all variables are active, the optimizer reduces drag by 15 counts (i.e., 5%) compared with the baseline design.

Future work will focus on the application of the framework to optimize unconventional configurations, such as truss-braced wings. It is also necessary to investigate methods to reduce the noise associated with changes in the overset connectivity, such as switching to a frozen connectivity approach at the end of the optimization.

6 Acknowledgments

This work was partially funded by NASA through award No. NNX14AC73A–Technical Monitor Tristan Hearn. The first author acknowledges support from the Brazilian Air Force, and the second author acknowledges support from the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1256260.

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562.

References

[1] Tétrault, P.-A., Schetz, J. A., and Grossman, B., "Numerical Prediction of Interference Drag of Strut-surface Intersection in Transonic Flow," *AIAA Journal*, Vol. 39, No. 5, 2001, pp. 857–864.

[2] Duggirala, R. K., Roy, C. J., and Schetz, J. A., "Analysis of Interference Drag for Strut-strut Interaction in Transonic Flow," *AIAA Journal*, Vol. 49, No. 3, 2011, pp. 449–462.

[3] Gur, O., Bhatia, M., Schetz, J. A., Mason, W. H., Kapania, R. K., and Mavris, D. N., "Design Optimization of a Truss-braced-wing Transonic Transport Aircraft," *Journal of Aircraft*, Vol. 47, No. 6, 2010. doi:10.2514/1.47546.

[4] Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, No. 3, Sept. 1988, pp. 233–260.

[5] 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, March 2010, pp. 373–391. doi:10.1016/j.compfluid.2009.09.013.

[6] Ivaldi, D., Secco, N. R., Chen, S., Hwang, J. T., and Martins, J. R. R. A., "Aerodynamic Shape Optimization of a Truss-Braced-Wing Aircraft," *Proceedings of the 16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Dallas, TX, June 2015. doi:10.2514/6.2015-3436.

[7] Steger, J. L., Dougherty, F. C., and Benek, J. A., "A Chimera Grid Scheme. Multiple Overset Body-conforming Mesh System for Finite Difference Adaptation to Complex Aircraft Configurations," *Advances in grid generation; Proceedings of the Applied Mechanics, Bioengineering, and Fluids Engineering Conference*, American Society of Mechanical Engineers, Houston, TX, 1983, pp. 59–69, A84-11576 02-64.

[8] Chan, W. M. and Buning, P. G., "Surface Grid Generation Methods for Overset Grids," *Computers & fluids*, Vol. 24, No. 5, 1995, pp. 509–522.

[9] Chan, W. M. and Steger, J. L., "Enhancements of a Three-dimensional Hyperbolic Grid Generation Scheme," *Applied Mathematics and Computation*, Vol. 51, No. 2, 1992, pp. 181–205.

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

[11] Hahn, A., "Vehicle Sketch Pad: a Parametric Geometry Modeler for Conceptual Aircraft Design," *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2010. doi:10.2514/6.2010-657, AIAA 2010-657.

[12] Hwang, J. T. and Martins, J. R. R. A., "GeoMACH: Geometry-centric MDAO of Aircraft Configurations with High Fidelity," *Proceedings of the 14th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Indianapolis, IN, Sept. 2012. doi:10.2514/6.2012-5605, AIAA 2012-5605.

[13] Haimes, R. and Drela, M., "On the Construction of Aircraft Conceptual Geometry for High-fidelity Analysis and Design," *50th AIAA Aerospace sciences meeting including the new horizons forum and aerospace exposition*, 2012. doi:10.2514/6.2012-683, AIAA 2012-0683.

[14] Bobrowski, K., Ferrer, E., Valero, E., and Barnewitz, H., "Aerodynamic Shape Optimization Using Geometry Surrogates and Adjoint Method," *AIAA Journal*, Vol. 55, No. 10, 2017, pp. 3304–3317. doi:10.2514/1.J055766.

[15] Sederberg, T. W. and Parry, S. R., "Free-form Deformation of Solid Geometric Models," *SIGGRAPH Comput. Graph.*, Vol. 20, No. 4, Aug. 1986, pp. 151–160. doi:10.1145/15886.15903.

[16] 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*, No. AIAA 2010-9231, Fort Worth, TX, Sept. 2010. doi:10.2514/6.2010-9231.

[17] Samareh, J. A., "Survey of Shape Parameterization Techniques for High-Fidelity Multidisciplinary Shape Optimization," *AIAA Journal*, Vol. 39, No. 5, 2001, pp. 877–884.

[18] Bonet, J. and Peraire, J., "An Alternating Digital Tree (ADT) Algorithm for 3D Geometric Searching and Intersection Problems," *International Journal for Numerical Methods in Engineering*, Vol. 31, No. 1, 1991, pp. 1–17.

[19] Möller, T., "A Fast Triangle-triangle Intersection Test," *Journal of Graphics Tools*, Vol. 2, No. 2, 1997, pp. 25–30.

[20] Eriksson, L., "Generation of Boundary-conforming Grids Around Wing-body Configurations Using Transfinite Interpolation," *AIAA Journal*, Vol. 20, No. 10, 1982, pp. 1313–1320.

[21] Hascoët, L. and Pascual, V., "The Tapenade Automatic Differentiation tool: Principles, Model, and Specification," *ACM Transactions On Mathematical Software*, Vol. 39, No. 3, 2013.

[22] Anderson, E., Bai, Z., Dongarra, J., Greenbaum, A., McKenney, A., Du Croz, J., Hammarling, S., Demmel, J., Bischof, C., and Sorensen, D., "LAPACK: A Portable Linear Algebra Library for High-performance Computers," *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing*, Supercomputing '90, IEEE Computer Society Press, Los Alamitos, CA, USA, 1990, pp. 2–11.

[23] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D., *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 3rd ed., 1999.

[24] Giles, M. B., "Collected Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation," *Advances in Automatic Differentiation*, Springer, 2008, pp. 35–44.

[25] Hascoët, L., Vázquez, M., and Dervieux, A., "Automatic Differentiation for Optimum Design, Applied to Sonic Boom Reduction," *Computational Science and Its Applications - ICCSA 2003*, 2003, pp. 976–976. doi:10.1007/3-540-44843-8_10.

[26] Lyu, Z., Xu, Z., and Martins, J. R. R. A., "Benchmarking Optimization Algorithms for Wing Aerodynamic Design Optimization," *Proceedings of the 8th International Conference on Computational Fluid Dynamics*, Chengdu, Sichuan, China, July 2014, ICCFD8-2014-0203.

[27] Yu, Y., Lyu, Z., Xu, Z., and Martins, J. R. R. A., "On the Influence of Optimization Algorithm and Starting Design on Wing Aerodynamic Shape Optimization," *Aerospace Science and Technology*, 2018. doi:10.1016/j.ast.2018.01.016, (In press).

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

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

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

[31] 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, January 2016, pp. 276–293. doi:10.2514/1.C033328.

[32] Brooks, T. R., Kennedy, G. J., and Martins, J. R. R. A., "High-fidelity Multipoint Aerostructural Optimization of a High Aspect Ratio Tow-steered Composite Wing," *Proceedings of the 58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech Forum*, Grapevine, TX, January 2017. doi:10.2514/6.2017-1350.

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

[34] van der Weide, E., Kalitzin, G., Schluter, J., and Alonso, J. J., "Unsteady Turbomachinery Computations Using Massively Parallel Platforms," *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 2006, AIAA 2006-0421.

[35] 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, April 2008, pp. 863–873. doi:10.2514/1.29123.

[36] Klopfer, G., Hung, C., Van der Wijngaart, R., and Onufer, J., "A Diagonalized Diagonal Dominant Alternating Direction Implicit (D3ADI) Scheme and Subiteration Correction," *29th AIAA, Fluid Dynamics Conference*, 1998. doi:10.2514/6.1998-2824, AIAA 1998-2824.

[37] Jameson, A., "Analysis and Design of Numerical Schemes for Gas Dynamics, 1: Artificial Diffusion, Upwind Biasing, Limiters and Their Effect on Accuracy and Multigrid Convergence," *International Journal of Computational Fluid Dynamics*, Vol. 4, No. 3-4, 1995, pp. 171–218. doi:10.1080/10618569508904524.

[38] Knoll, D. A. and Keyes, D. E., "Jacobian-free Newton–Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, Vol. 193, No. 2, 2004, pp. 357–397.

[39] Lee, Y. and Baeder, J. D., "Implicit Hole Cutting - A New Approach to Overset Grid Connectivity," *16th AIAA Computational Fluid Dynamics Conference, Fluid Dynamics and Co-located Conferences*, 2003. doi:10.2514/6.2003-4128, AIAA 2003-4128.

[40] 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.

[41] Chan, W. M., "Enhancements to the Hybrid Mesh Approach to Surface Loads Integration on Overset Structured Grids," *19th AIAA Computational Fluid Dynamics, Fluid Dynamics and Co-located Conferences*, 2009. doi:10.2514/6.2009-3990, AIAA 2009-3990.

[42] 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*, Grapevine, TX, January 2017. doi:10.2514/6.2017-0357.

[43] Tinoco, E. N., Brodersen, O., Keye, S., and Laflin, K., "Summary of Data from the Sixth AIAA CFD Drag Prediction Workshop: CRM Cases 2 to 5," *55th AIAA Aerospace Sciences Meeting*, 2017. doi:10.2514/6.2017-1208, AIAA 2017-1208.

[44] Spalart, P. R. and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," *30th Aerospace Sciences Meeting and Exhibit*, 1992. doi:10.2514/6.1992-439, AIAA 1992-439.

[45] Coder, J. G., Pulliam, T. H., Hue, D., Kenway, G. K., and Sclafani, A. J., "Contributions to the 6th AIAA CFD Drag Prediction Workshop Using Structured Grid Methods," *AIAA SciTech Forum*, American Institute of Aeronautics and Astronautics, Jan. 2017. doi:10.2514/6.2017-0960.

[46] Dacles-Mariani, J., Zilliac, G. G., Chow, J. S., and Bradshaw, P., "Numerical/Experimental Study of a Wingtip Vortex in the Near Field," *AIAA Journal*, Vol. 33, No. 9, 1995, pp. 1561–1568.

[47] Spalart, P. R., "Strategies for Turbulence Modelling and Simulations," *International Journal of Heat and Fluid Flow*, Vol. 21, No. 3, 2000, pp. 252–263.

[48] Lyu, Z., Kenway, G. K., Paige, C., and Martins, J. R. R. A., "Automatic Differentiation Adjoint of the Reynolds-Averaged Navier–Stokes Equations with a Turbulence Model," *21st AIAA Computational Fluid Dynamics Conference*, San Diego, CA, Jul. 2013. doi:10.2514/6.2013-2581.

[49] Gill, P., Murray, W., and Saunders, M., "SNOPT: An SQP Algorithm for Large-scale Constrained Optimization," *SIAM Journal on Optimization*, Vol. 12, No. 4, 2002, pp. 979–1006. doi:10.1137/S1052623499350013.

[50] 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, November 2013, pp. 2582–2599. doi:10.2514/1.J052184.

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

[52] Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F., "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries," *Modern Software Tools in Scientific Computing*, edited by E. Arge, A. M. Bruaset, and H. P. Langtangen, Birkhäuser Press, 1997, pp. 163–202.

[53] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H., "PETSc Web page," `http://www.mcs.anl.gov/petsc`, 2016.

[54] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H., "PETSc Users Manual," Tech. Rep. ANL-95/11 - Revision 3.7, Argonne National Laboratory, 2016.

[55] 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.

[56] Laflin, K. R., Klausmeyer, S. M., Zickuhr, T., Vassberg, J. C., Wahls, R. A., Morrison, J. H., Brodersen, O. P., Rakowitz, M. E., Tinoco, E. N., and Godard, J.-L., "Data Summary from Second AIAA Computational Fluid Dynamics Drag Prediction Workshop," *Journal of Aircraft*, Vol. 42, No. 5, 2005, pp. 1165–1178.

[57] Vassberg, J., Tinoco, E., Mani, M., Brodersen, O., Eisfeld, B., Wahls, R., Morrison, J., Zickuhr, T., Laflin, K., and Mavriplis, D., "Summary of the Third AIAA CFD Drag Prediction Workshop," *45th AIAA Aerospace Sciences Meeting and Exhibit*, 2007. doi:10.2514/6.2007-260, AIAA 2007-260.

[58] Lee, B. J., Yim, J. W., Yi, J. S., and Kim, C., "Aerodynamic Shape Optimization Using Discrete Adjoint Formulation Based on Overset Mesh Technique," *KSAS International Journal*, Vol. 8, No. 1, 2007, pp. 95–104.

[59] Brezillon, J. and Dwight, R. P., "Applications of a Discrete Viscous Adjoint Method for Aerodynamic Shape Optimisation of 3D Configurations," *CEAS Aeronautical Journal*, Vol. 3, No. 1, 2012, pp. 25–34.

[60] Xu, S., Timme, S., Mykhaskiv, O., and Müller, J.-D., "Wing-body Junction Optimisation with CAD-based Parametrisation Including a Moving Intersection," *Aerospace Science and Technology*, Vol. 68, 2017, pp. 543–551.

[61] Vassberg, J., Sclafani, A., and DeHaan, M., "A Wing-body Fairing Design for the DLR-F6 Model: a DPW-III Case Study," *23rd AIAA Applied Aerodynamics Conference*, 2005. doi:10.2514/6.2005-4730, AIAA 2005-4730.

[62] 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, April 2015, pp. 968–985. doi:10.2514/1.J053318.

[63] ANSYS ICEM CFD 14.0, ANSYS, Inc., 275 Technology Drive, Canonsburg, PA, October 2011.