Flexible Formulation of Spatial Integration Constraints in Aerodynamic Shape Optimization

Benjamin J. Brelje
Joshua L. Anibal
Anil Yildirim
Charles A. Mader
Joaquim R.R.A. Martins
University of Michigan Department of Aerospace Engineering

Abstract In aircraft design, spatial integration places limits on aerodynamic and structural performance. While the outer mold line shape largely determines aircraft aerodynamic characteristics, aircraft systems and passengers or payloads must fit inside. Emerging categories of aircraft, such as electric aircraft, are likely to experience new and critical spatial integration challenges. Existing aerodynamic shape optimization tools can accept a narrowly defined set of geometric constraints. However, no known aerodynamic optimization framework can handle spatial integration constraints derived directly from 3D geometry. We propose a general geometric constraint formulation based on triangulated 3D geometry of both the outer mold line surface and the “object(s) to fit.” The constraint consists of two metrics: the length of the intersection curve(s) between any two objects, and the Kreisselmeier–Steinhauser function aggregated minimum distance. Our implementation of the intersection and distance calculations is fast and analytically differentiable with respect to geometric design variables, making it suitable for efficient gradient-based optimization. We validate our constraint formulation on three RANS-based aerodynamic shape optimization problems: a 2D fairing design problem, a 3D fairing design problem, and the design of an aeroshell to surround a human avatar. We show that this approach is robust and efficient, which enables the aerodynamic optimization of bodies containing objects with arbitrary shapes.

Nomenclature

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>spatial integration tolerance parameter</td>
</tr>
<tr>
<td>$d_{min}$</td>
<td>minimum distance</td>
</tr>
<tr>
<td>$k$</td>
<td>number of objects to fit</td>
</tr>
<tr>
<td>$n$</td>
<td>number of surface mesh triangles</td>
</tr>
<tr>
<td>$m$</td>
<td>number of object mesh triangles</td>
</tr>
<tr>
<td>$p$</td>
<td>number of geometric design variables</td>
</tr>
<tr>
<td>$x$</td>
<td>geometric design variables</td>
</tr>
<tr>
<td>$C_D$</td>
<td>drag coefficient</td>
</tr>
<tr>
<td>$L$</td>
<td>length of intersection curve(s)</td>
</tr>
<tr>
<td>$V_{j(1,2,3)}$</td>
<td>triangle vertices of the $j$th triangle</td>
</tr>
<tr>
<td>ASO</td>
<td>aerodynamic shape optimization</td>
</tr>
<tr>
<td>CFD</td>
<td>computational fluid dynamics</td>
</tr>
<tr>
<td>KS</td>
<td>Kreisselmeier–Steinhauser</td>
</tr>
<tr>
<td>MDO</td>
<td>multidisciplinary design optimization</td>
</tr>
<tr>
<td>MIP</td>
<td>mixed-integer programming</td>
</tr>
<tr>
<td>OML</td>
<td>outer mold line</td>
</tr>
</tbody>
</table>
1 Introduction

Aerodynamic shape optimization (ASO) has become an increasingly popular tool for aircraft design [1, 2]. Multidisciplinary design optimization (MDO) practitioners have various choices for aerodynamic models to suit their requirements, ranging from low-fidelity panel codes to Reynolds-averaged Navier–Stokes (RANS) simulations. Aerostructural coupling has also evolved from 1D beam models to full aeroelastic finite element models [3, 4]. Recent interest in boundary layer ingestion has led to close coupling of aerodynamic and propulsion models in aeropropulsive design optimization, e.g. [5, 6].

While the “hard” aerosciences have been incorporated into increasingly high-fidelity MDO frameworks, one crucial aspect of aircraft design has been left out: spatial integration. Spatial integration, at a minimum, requires that all crew, passengers, payloads, systems components, and energy storage fit within the aircraft outer mold line (OML), and that objects do not overlap with each other. Spatial integration constraints directly drive important design features found on aircraft today; from wing-to-body fairing depth on commercial aircraft, to carefully-shaped actuator “bumps” on the wings of stealth military aircraft. Recent electric and hybrid-electric propulsion concepts force hard decisions about where to locate hundreds or thousands of pounds of batteries that take much more volume than conventional fuel. For example, Kitty Hawk’s Cora eVTOL flight test demonstrator locates the battery behind the cabin within the fuselage upsweep, while Uber Technologies’ eCRM-003 uses large booms reminiscent of fighter external fuel tanks [7]. Since range is closely related to the quantity of batteries that can be carried on board, trade studies for electric aircraft need to rigorously account for the aircraft volume growth necessary to accommodate battery packs of varying sizes. This is a closely-coupled geometry-aerodynamic-structural-performance cycle, which should be analyzed through MDO. Safety considerations may introduce geometric requirements beyond non-intersection. For example, transport aircraft are required to physically separate critical systems components so no single failure can result in the loss of redundant functions. These kinds of trade studies are not easily accomplished using state-of-the-art MDO frameworks and present a barrier to the industrial adoption of MDO in practical vehicle design.

This paper starts with a review of existing geometric integration (or “packing”) optimization approaches, both from aerospace MDO and from outside the aerospace literature. Next, we develop a mathematical basis for an MDO geometric constraint, followed by a description of our computational implementation. Finally, we demonstrate aerodynamic shape optimization subject to spatial integration on three test cases of increasing complexity.

1.1 Spatial Constraints in Existing MDO Frameworks

Several simple methods for imposing geometric constraints on shape optimization problems have already been developed. Generally, the methods involve computing point thicknesses at specified locations on the aircraft OML. Kenway et al. [8] developed pyGeo, a geometry engine for high-fidelity MDO, which provides the following geometric constraints:

Point thickness constraints — the simplest means of preventing the optimizer from reducing thickness too much in one spot.

1D thickness constraints — enforced using a line of point thickness constraints suitable for preserving thickness along a line; for example, spar depth.

2D thickness constraints — enforced using an array of point thickness constraints suitable for preserving thickness over an area; for example, wing box thickness.

Volume constraints — useful for ensuring that a wing design has sufficient space for fuel.

Although these constraints have been useful in a variety of wing and aircraft design optimization problems [4, 9, 10], they do not provide sufficient freedom to accurately reflect complex geometries. In theory, relatively complex shapes can be represented by imposing a large number of point thickness constraints, but there are important practical limitations. An MDO engineer would need to convert a 3D model of the object-to-fit into point thickness constraints by hand, which is error-prone and laborious. If the shape optimization involves both planform and thickness variables, the constrained shape will stretch and deform in unpredictable ways compared to the original intent.
Using thickness constraints to represent complex geometry also limits design freedom. If a thickness constraint is imposed at a planform location, the object-to-fit is effectively locked into the same planform location during the entire optimization. Unless the starting location is well-chosen, the lack of positioning design variables will cause the resulting shape to be suboptimal. We can also easily envision realistic design problems in which components change shape as design requirements change (e.g., motor diameter with rated torque). Geometric constraints available today are not capable of handling these cases.

1.2 Packing Optimization Problems in Other Fields

While spatial integration has not been a focus in aircraft MDO, the operations research (OR) field has developed approaches suitable to the optimal packing of objects into a specified volume. “Tetris-like” rectangular volumes may be efficiently packed into a simple container using a mixed-integer programming (MIP) approach [11]. This approach allows 90-degree permutations but not arbitrary rotations. It also requires a fixed outer bounding volume shape, which is analogous to an aircraft OML. There is no existing provision for coupling together the MIP problem with physics-based optimization (e.g., for minimum drag) of the outer volume. The MIP Tetris-like-solid formulation has been applied to spacecraft design applications [11].

The trunk packing problem is a canonical problem in the operations research packing optimization literature [12]. Automotive industry standards exist for the type and quantity of objects (for example, suitcases and boxes) that must be able to fit into an automobile trunk. It is advantageous for auto manufacturers to fit as many of these irregular objects as possible into the trunk’s bounded volume. Many approaches have been proposed to tackle this problem. One of the most general approaches is to use a volumetric discretization of the trunk; the subdivisions are called voxels [13, 14]. Using a genetic algorithm, irregular objects can be rotated and translated in six degrees of freedom until no voxel is occupied by more than one object [14]. This approach is very general and has been extended to handle a multiobjective engine compartment optimization to explore the tradeoff between compactness and maintainability [12]. Voxelization can also handle optimization problems where the components being packed vary in size during the optimization, such as maximizing coolant tank volume while still fitting all required components in the engine compartment [12].

However, like the MIP formulation, the volume discretization depends on a fixed OML, rendering it unsuitable for aerodynamic shape optimization applications where the OML changes during optimization. The presence-or-absence voxel test is binary and non-differentiable, which is not suitable for gradient-based methods. Gradient-based methods are a necessity for high-fidelity shape optimization because gradient-free methods scale poorly with increasing degrees of design freedom and result in intractable computational cost [15].

Gradient-based packing optimization has been explored, but not as extensively as gradient-free approaches. Stoyan et al. [16] demonstrated gradient-based packing optimization of simple 2D and 3D geometric shapes. The approach depends on calculating a form of distance metric the authors call quasi-φ functions. These functions can be derived analytically for some classes of simple geometric shapes (such as a sphere or cone). Using this approach is desirable because the quasi-φ functions can be differentiated, making them suitable for gradient-based optimization.

Unfortunately, the quasi-φ approach is not that general because it applies only to classes of geometric shapes for which the functions can be derived. This may be suitable for some conceptual design scenarios, where hold-out volumes of general shape and volume are known. For aircraft design, using this approach would require representing the OML geometry of the aircraft as a composition of geometric surfaces with known φ functions, which would require a major research effort in itself. The mathematics underlying the approach is also likely to be unfamiliar to most aerospace engineering practitioners. However, the results demonstrate that gradient-based packing optimization may be promising in at least some applications.

While the OR field has made substantial progress towards solving packing optimization problems, none of the prior approaches are suitable for use with high-fidelity MDO in aerospace. A new geometric constraint formulation is required, with the following properties:

**General** —The method should be able to represent arbitrary surfaces (including, at a minimum, a wing surface with high fidelity). The constraint metric calculation must not depend on the convexity of each object (since aircraft wings are often locally concave).

**Differentiable** —The constraint metric(s) must be differentiable and at least $C^0$ continuous (preferably
Efficient — The constraint metric(s) must have computation time and memory requirements with acceptable scaling properties (with hundreds to tens of thousands of geometric design variables, objects, and surface polygons).

2 Deriving a Mathematical Definition of Spatial Integration for MDO

To derive a mathematical definition of containment, let us consider some component (with outer surface $A$) to be fit inside OML surface $B$ (Figure 1).

![Figure 1: Schematic view of a wing OML and interior component](image)

**Definition.** Let $A$ be a connected surface defined in three-dimensional real space.

**Definition.** Let $B$ be a closed, connected, orientable surface defined in three-dimensional real space.

Closed orientable 3D surfaces have a defined interior volume and can be thought of as 3D solids or closed 3D shells. A watertight CFD mesh always meets this definition. Formally, the volume enclosed by $B$ and the surface $B$ are not identical, but we will call them both $B$ for simplicity.

**Definition.** Let $d_{\text{min}}$ be the minimum distance between $A$ and $B$.

**Definition.** Let $L$ be the sum of the length(s) of the intersection curve(s) between $A$ and $B$, or 0 if no intersection.

Next, we provide an intuitive definition of geometric containment.

**Definition.** A is contained within $B$ if and only if all points $P$ on $A$ are inside $B$.

Testing an infinite number of points $P$ is not computationally tractable. Instead, we can replace this definition of containment with an equally intuitive one.

**Axiom.** A is contained within $B$ if and only if:

1. any single point $P$ on $A$ is inside $B$, and
2. $A$ and $B$ do not intersect.

We further develop the definition of enclosure as a composition of computable tests.

**Lemma.** If two surfaces $A$ and $B$ intersect, $d_{\text{min}} = 0$.

**Corollary.** If $d_{\text{min}} > 0$, $A$ and $B$ are nonintersecting.

Finally, we arrive at the mathematical definition of our spatial integration constraint.

**Theorem.** Surface $A$ is contained within bounding surface $B$ if and only if:

1. any single point $P$ on $A$ is inside $B$, and
2. the minimum distance $d_{\text{min}}$ between $A$ and $B > 0$.

Our work focuses on Condition 2—the minimum distance tests between $A$ and $B$. In an optimization context, Condition 1 (at least one enclosed point) can be satisfied by setting reasonable bounds on the geometric design variables. Alternatively, if there is a chance that the interior object might “escape” from the bounding volume during optimization, a ray-casting test can be used instead [17], as follows:
**Theorem.** Point \( P \) is inside \( B \) if and only if a ray, originating from \( P \) and traveling in an arbitrary direction, intersects \( B \) an odd number of times.

Unfortunately, the one-sided nature of Euclidean distance presents a practical difficulty for gradient-based optimizers because the value (and more importantly, the gradient) of \( d_{\text{min}} \) goes to zero once \( A \) and \( B \) intersect. Figure 2 illustrates the minimum distance as a sphere translates closer to the edge of an ellipsoidal bounding volume. If the surfaces are discretized for computation (for example, using a triangulated surface), an additional problem arises as \( d_{\text{min}} \) between the facets never reaches machine zero, even while intersected.

![Figure 2: The minimum distance goes to zero post-intersection, presenting optimization difficulties.](image)

To help the optimizer find its way back into feasible space once intersection occurs, we include the total length \( L \) of the intersection curve(s) as an additional constraint. The intersection curve length provides feasibility gradient information once shapes are intersected and ensures that intersected surfaces are not falsely assessed as feasible due to discretization error (Figure 2).

**Claim.** A gradient-based optimizer can ensure that convex component \( A \) fits inside convex bounding surface \( B \) if an arbitrary point \( P \) on \( A \) remains inside \( B \), and the following nonlinear constraints are imposed:

1. \( d_{\text{min}} > 0 \)
2. \( L = 0 \)

If design variable bounds and the geometric parameterization allow \( A \) to completely “escape” \( B \) (e.g., the fourth panel of Figure 2), the optimizer should be configured to reject iterations where \( P \) is outside \( B \) using a binary test such as ray tracing.

A further complication occurs for nonconvex geometries. If the optimizer takes too large of a step into the intersected region, a gradient-descent or similar strategy will not allow the optimizer to return to the feasible region, as illustrated in Figure 3. Therefore, the optimizer must not take steps into the infeasible region that are too large.

**Claim.** A gradient-based optimizer can ensure that component \( A \) fits inside bounding surface \( B \) if an arbitrary point \( P \) on \( A \) remains inside \( B \), the optimizer does not step into an intersected region, and the following nonlinear constraints are imposed:

1. \( d_{\text{min}} > 0 \)
2. \( L = 0 \)

With this final statement, we have derived a mathematical definition of geometric feasibility that is computable, \( C^0 \) continuous, differentiable, and qualitatively well-behaved for nonlinear optimization algorithms.
Figure 3: The optimizer can get stuck in the infeasible region for nonconvex geometries.

Figure 4: Triangle distance tests

(a) Point-triangle tests (6 of 6 shown)  (b) Edge-edge tests (3 of 9 shown; repeat for each edge)

3 Computing Geometric Constraints

To compute the mathematical functions derived above, we needed to choose a data structure to represent the surface geometry. While there are numerous approaches to computational geometry, we chose simple triangulated surfaces for generality and efficiency. Many unstructured CFD grids are natively triangular due to tetrahedral volumes, and structured grids using hexahedral volumes can be easily triangulated. Spline or other parametric surfaces are also easily triangulated. Furthermore, many efficient distance and intersection algorithms have already been developed for triangular geometry primitives.

The minimum distance between two triangles can be found through six point-triangle distance tests and nine edge-edge tests (Figure 4). We implemented the point-triangle and edge-edge distance tests of Ericson [18]. While lower-cost distance tests exist in the literature, Ericson’s approach is vectorizable, allowing us to use analytic differentiation to obtain derivatives.

During optimization, the pair of triangles determining \( d_{\min} \) changes for every iteration, and the gradients of \( d_{\min} \) are discontinuous. Tracking \( d_{\min} \) alone also ignores useful information from the (possibly many) pairs of triangles that are almost the closest. Therefore, constraining \( d_{\min} \) by itself is likely to produce bad results. Alternatively, all of the edge-edge and point-triangle distances could be constrained > 0. However,
problem scaling makes this impractical. Let us envision a problem with \( n = 10^4 \) OML mesh facets, \( m = 10^3 \) mesh facets per object, \( k = 10^4 \) objects, and \( p = 10^2 \) geometric design variables. There are \( O(nmpk) = 10^{10} \) gradient entries to store, which easily exceeds available memory even for modest problems.

Ideally, all of the almost closest facets would contribute to a single constraint. A good way of achieving this is constraint aggregation using the alternative form of the Kreisselmeier–Steinhauser (KS) function [19],

\[
KS[\mathbf{g}(\mathbf{x})] = g_{\text{max}}(\mathbf{x}) + \frac{1}{\rho} \ln \left( \sum_{j=1}^{m} e^{\rho (g_j(\mathbf{x}) - g_{\text{max}}(\mathbf{x}))} \right)
\]

(1)

where \( \mathbf{g}(\mathbf{x}) \leq 0 \) is the vector of inequality constraints, evaluated at the design point \( \mathbf{x} \), \( g_{\text{max}}(\mathbf{x}) \) is the maximum constraint value at the current design point, and \( \rho \) is a constant associated with the aggregation function. For spatial integration, the constraint vector \( \mathbf{g}(\mathbf{x}) = -d_j(\mathbf{x}) \), where \( d_j(\mathbf{x}) \) is the vector of all distance test results between facets of \( A \) and \( B \). Using this relationship, we obtain the final form of the distance constraint for optimization:

\[
KS_{\text{geom}}(\mathbf{x}) = \frac{1}{\rho} \ln \left( \sum_{j=1}^{J} e^{\rho (d_{\text{min}}(\mathbf{x}) - d_j(\mathbf{x}))} \right) - d_{\text{min}}(\mathbf{x}) \leq 0,
\]

(2)

where \( J = 15mn \), the number of distance tests between \( A \) and \( B \).

The KS function produces a conservative estimate of the maximum value of its inputs, while preserving function smoothness and derivative information from almost-active inputs. As the minimum distance between the two objects approaches zero, the KS function returns a positive (infeasible) value to the optimizer. The level of conservativeness of the estimate can be adjusted by changing \( \rho \). Higher \( \rho \) values lead to a tighter spatial fit, but the problem may be more poorly conditioned. As \( \rho \to \infty \), \( KS_{\text{geom}} \to -d_{\text{min}} \), but information from other close pairs of triangles is lost.

The second constraint (intersection length) is calculated following the method of Möller [20]. First, intersections between pairs of triangles are detected, if any. Then, the line segment(s) \( S_i \) of intersection are computed. Finally, total length \( L \) is obtained by summing all intersection line segment lengths.

The minimum distances and intersections between two surfaces consisting of \( n \) and \( m \) triangles, respectively can be computed by performing \( O(nm) \) pairwise distance tests. This operation is embarrassingly parallel, since none of the triangle-triangle comparisons depend on any of the others. Because \( n \) and \( m \) are both on the order \( 10^3–10^5 \), the distance tests must be very efficient.

To obtain the desired computational performance, we considered using a compiled language (such as FORTRAN) and incorporating branch logic with many early exits to avoid unnecessary computations. We would then obtain gradients for optimization via an automatic differentiation (AD) tool. An alternative approach for parallel problems is to use a graphics processing unit (GPU). GPUs are so much faster than CPUs for repetitive parallel tasks that wasted computations may be acceptable and the intricate branch logic to avoid them may not be necessary. If distance comparison between millions of triangles can be posed as vector or tensor operations on GPUs, then the code can be differentiated analytically.

Researchers have avoided using GPUs in the past partly because of the steep learning curve. Fortunately, the maturation of machine learning frameworks has made GPU computing substantially more accessible. TensorFlow is an open-source Python package originally developed at Google for machine learning workflows [21]. Since the majority of production machine learning tasks are performed on specialized hardware rather than CPUs, TensorFlow includes a Python API providing access to fast mathematical routines on GPUs. We have verified speedups of 25–100 for typical vector and tensor operations (such as addition, dot product, and min/max) compared to numpy with Intel MKL libraries on CPUs. Another compelling feature of TensorFlow is the built-in gradient computation. All mathematical operations and most control flow operations have reverse-mode differentiated code built in. TensorFlow propagates reverse-mode derivatives of model outputs with respect to model inputs across the graph of operations. In “eager mode”, the “GradientTape” object provides analytic derivatives at runtime.

We implemented the mathematical approach described in Section 2 as a Python library under the working title GeoGrad. We chose to use a vectorized approach for the computations. TensorFlow is feature rich, but for this project, we are essentially using it as “numpy for GPUs”. We used the GradientTape capability to compute derivatives of \( KS_{\text{geom}} \) and \( L \) with respect to the vertices of the input surfaces \( V_i \). Only a few additional lines of code were required to obtain derivatives once the constraint function was written.
4 Aerodynamic Shape Optimization Subject to Geometric Constraints

To validate our general geometric integration constraint, we constructed three aerodynamic shape optimization test cases of increasing complexity: a 2D fairing around a box for minimum drag, a 3D fairing around a cylinder for minimum drag, and a 3D fairing around a human avatar for minimum drag.

4.1 Methodology

We performed all optimizations using the MDO of Aircraft Configurations in High Fidelity (MACH) framework [1, 3]. MACH is a set of Python-, FORTRAN-, and C++-based tools and utilities that provide all necessary geometry, aerodynamics, and structural analyses to perform high-fidelity aircraft MDO. Individual components of MACH are described in detail in previous publications [1, 22], but we briefly review the overall methodology here.

For aerodynamic analysis, we used ADflow, which solves the RANS equations on structured overset meshes (though we did not use overset capability for this study). We used ADflow to evaluate drag and its sensitivities. The adjoint solver in ADflow [23] computes the gradients of output quantities (e.g., drag) efficiently, even for problems with thousands of design variables. Together, ADflow and the MACH framework have been used to perform transonic wing optimization for minimum drag [1, 24–26], as well as aerostructural optimization [4, 9, 27].

Unlike wing optimization cases, the test cases for this paper often go through intermediate designs with bluff-body aerodynamics during optimization. This primarily occurs due to the cross-flow conditions we include in our multipoint aerodynamic problem. Many RANS solvers have trouble converging cases with separated flow. However, ADflow is particularly robust in solving these intermediate cases.

ADflow’s robustness is attributable to the approximate Newton–Krylov (ANK) nonlinear solver scheme [28]. The ANK solver uses a matrix-free approximate Jacobian formulation with the pseudo-transient continuation method. Using ADflow with ANK, we can converge cases with heavy separation in the flow-field, improving the likelihood that the optimization will succeed. Intermediate steps with difficult aerodynamic characteristics can also be avoided by utilizing trust region methods, by carefully re-formulating the optimization problem into multiple subproblems, or both. However, both of these approaches require intervention by an expert user. With ANK, the user need not be as careful with problem setup. This enabled us to rapidly study the range of challenging optimization problems we present in this paper.

Two other MACH tools, pyGeo and IDWarp, provide geometric parameterization and mesh deformation, respectively. pyGeo uses a free-form deformation (FFD) approach to modify the surface mesh with respect to the design variables [29]. The FFD approach embeds surface mesh vertices into a solid block, which deforms as geometric design variables change during optimization. Since only the surface mesh points are directly parameterized, changes in the surface mesh must be propagated to the volume mesh. IDWarp accomplishes this mesh deformation using an inverse distance mesh warping method similar to the method developed by Luke et al. [30]. The initial volume mesh itself is generated using pyHyp [22], which uses a hyperbolic volume mesh marching scheme [31].

pyGeo also contains a submodule, DVConstraints, which computes nonlinear geometric constraints, such as thickness and volume constraints, based on the surface mesh geometry. We extended DVConstraints with GeoGrad to compute the general 3D geometry constraint described in Section 2.

Finally, we used SNOPT (Sparse Nonlinear OPTimizer) [32], wrapped in pyOptSparse [33], as the nonlinear optimizer to drive the problem. SNOPT is especially useful for large-scale problems with many design variables, and for problems with functions that are expensive to evaluate. It has been used repeatedly for aerodynamic and aerostructural optimization problems [1, 25]. A key feature of SNOPT for this study is its tolerance of objective function evaluation failures. If an analysis code fails (such as a CFD convergence failure), SNOPT reduces its step size and tries again. We used this feature to handle intersected cases. If the geometry tool detects intersection between the CFD surface mesh and the constraint object mesh, we return a failure flag to SNOPT, which will force the optimizer to “back up” and try again with a less aggressive step. This worked qualitatively well for us, since it avoids wasting an expensive CFD evaluation.

We ran all optimization cases on a desktop computer with an Intel Core i7-6700K CPU (4 cores) and an NVIDIA GeForce GTX 970 GPU. ADflow runs in parallel on the CPU, while the geometric constraint framework runs on the GPU.
4.2 2D Aerodynamic Shape Optimization Around a Box

We first demonstrated the new geometry constraint using a 2D aerodynamic shape optimization problem. The objective function was to minimize multipoint average drag around a box at $0^\circ$ and $10^\circ$ crossflow angle. The starting surface mesh was a NACA 0012 airfoil (Figure 6). The 2D parameterization consisted of 22 FFD points providing fine shape control in the $y$ axis (Figure 5), one variable to translate the entire volume in the streamwise $x$ direction, and one variable for chord in the $x$-axis.

We imposed symmetry in the $y$-axis to effectively obtain a $-10^\circ$ crossflow case without running additional CFD cases. The optimization parameters are described in Table 1. To prevent negative warped mesh volumes near the trailing edge in single point cases, we also included a line of point thickness constraints at $x/c > 99\%$

Throughout the paper, $C_D$ is computed with respect to a fixed reference area of 1 m.

<table>
<thead>
<tr>
<th>Description</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multipoint average drag, or $C_D,\alpha=0$</td>
<td></td>
</tr>
<tr>
<td>Single point drag</td>
<td></td>
</tr>
<tr>
<td>$y$-axis FFD shape control variables</td>
<td>22</td>
</tr>
<tr>
<td>Streamwise chord length</td>
<td>1</td>
</tr>
<tr>
<td>Streamwise translation</td>
<td>1</td>
</tr>
<tr>
<td>Symmetry across $y$-axis</td>
<td>11</td>
</tr>
<tr>
<td>Intersection perimeter</td>
<td>1</td>
</tr>
<tr>
<td>Aggregated minimum distance</td>
<td>1</td>
</tr>
<tr>
<td>$M = 0.3$</td>
<td></td>
</tr>
<tr>
<td>$Re = 6 \times 10^6$</td>
<td></td>
</tr>
<tr>
<td>Sea level, standard day</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: 2D aerodynamic shape optimization parameters
We ran five cases with varying spatial tightness parameter $\rho$, with graphical results in Figure 7. Table 2 shows that as spatial tightness is increased, the potential drag reduction increases, with diminishing returns at large (tight) values of $\rho$. We also found that tightening spatial tolerances increased computational cost by about 50%. As $\rho$ increases, the local curvature (nonlinearity) of the function increases, which reduces the accuracy of the optimizer’s quasi-Newton step. A second effect of large $\rho$ is that the KS contribution from nearby facets becomes very large compared to distant segments. Floating point errors round the contributions of distant segments to zero in the constraint Jacobian, which becomes sparser as a result. Both of these effects degrade optimizer performance.

![Figure 7: 2D multipoint minimum drag results for varying spatial tolerances ($\rho$)](image)

<table>
<thead>
<tr>
<th>Case</th>
<th>Solution Time (min)</th>
<th>$N$ CFD evals</th>
<th>$C_D$</th>
<th>$\Delta C_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>–</td>
<td>–</td>
<td>0.01069</td>
<td>–</td>
</tr>
<tr>
<td>$\rho = 200$</td>
<td>130</td>
<td>40</td>
<td>0.01170</td>
<td>+9.4%</td>
</tr>
<tr>
<td>$\rho = 300$</td>
<td>131</td>
<td>42</td>
<td>0.01026</td>
<td>–4.0%</td>
</tr>
<tr>
<td>$\rho = 600$</td>
<td>123</td>
<td>56</td>
<td>0.00887</td>
<td>–17.0%</td>
</tr>
<tr>
<td>$\rho = 900$</td>
<td>185</td>
<td>80</td>
<td>0.00842</td>
<td>–21.2%</td>
</tr>
<tr>
<td>$\rho = 1200$</td>
<td>194</td>
<td>82</td>
<td>0.00820</td>
<td>–23.3%</td>
</tr>
</tbody>
</table>

Table 2: Drag reduction potential increases with tighter spatial integration tolerance $\rho$

### 4.3 3D Aerodynamic Shape Optimization Around a Simple Surface of Revolution

For a simple 3D test, we defined single-point ($0^\circ$) and multipoint ($0^\circ$ and $20^\circ$) drag minimization cases. We created a starting surface mesh consisting of a NACA 0012 airfoil revolved around the streamwise ($x$) axis (Figure 9). The structured surface and volume meshes consisted of 1802 and 237762 cells, respectively. The 3D parameterization consists of 192 FFD points which provide fine shape control along the $y$-axis (Figure 8a),
and an additional 17 parameters providing degrees of freedom in the $x$ and $z$ axes. We imposed symmetry in the crossflow ($x$-$y$) plane to effectively obtain a $-20^\circ$ crossflow case without running additional CFD cases. The optimization parameters are described in Table 3.

Figure 10 shows the optimized shape for the single point case. The drag decreased 29.3% compared to the baseline single point case. We see a long fairing with relatively tight leading edge curvature. In our experience, the tightly curved leading edge is characteristic of single point aerodynamic shape optimization since robustness to varying flow conditions is not required. Even though the problem is parameterized in a Cartesian frame, the finished shape is almost perfectly axisymmetric, as we would expect from a rotationally symmetric spatial constraint and flow condition.

Figure 11 shows the optimized shape for the multipoint case. Drag decreased 52.2% compared to the
Variable Description Quantity

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimize</td>
<td>avg($C_{D,\alpha=0}$, $C_{D,\alpha=20}$)</td>
<td>Multipoint average drag, or Single point drag</td>
</tr>
<tr>
<td>with respect to</td>
<td>$x_{FFDy}$</td>
<td>y-axis FFD shape control variables 192</td>
</tr>
<tr>
<td></td>
<td>$x_{chord}$</td>
<td>Streamwise chord length of “airfoil” slices along z-axis 8</td>
</tr>
<tr>
<td></td>
<td>$x_{translate}$</td>
<td>Streamwise translation of “airfoil” slices along z-axis 8</td>
</tr>
<tr>
<td></td>
<td>$x_{span}$</td>
<td>“Span” along z-axis 1</td>
</tr>
<tr>
<td>subject to</td>
<td>$x_{FFDy,\text{left}} = -x_{FFDy,\text{right}}$</td>
<td>Symmetry across y-axis 96</td>
</tr>
<tr>
<td></td>
<td>$L = 0$</td>
<td>Intersection perimeter 1</td>
</tr>
<tr>
<td></td>
<td>$KS_{\text{geom}} \leq 0$</td>
<td>Aggregated minimum distance 1</td>
</tr>
<tr>
<td>at condition</td>
<td>$M = 0.3$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$Re = 6 \times 10^6$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sea level, standard day</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: 3D aerodynamic shape optimization parameters

baseline multipoint case. We see a gently rounded nose and broader aft body closure angle, which visually matches the crossflow condition. Compared to the single point case, the crossflow condition has added thickness in the x-y plane and reduced length overall. The rounder nose and extra thickness-to-chord ratio both help improve resistance to flow separation at the crossflow condition. Visually, the x-y thickness is not required simply for spatial integration reasons, but the x-z plane generally tightly conforms to the cylinder. This illustrates that it is not evident a priori whether geometric constraints will be active for a particular problem; physics and geometry are strongly coupled.

Figure 11: 3D multipoint minimum drag result for cylinder, $\rho = 1200$

### 4.4 3D Aerodynamic Shape Optimization Around a Human Avatar

Finally, we set up single and multipoint optimization cases where the geometry to be enveloped is that of a person in a seated position, with parameters identical to the previous case (Table 3). We exported a high-resolution model of an average U.S. adult in a seated driving position to an .stl file using the University of Michigan Transportation Research Institute’s (UMTRI) online tool\(^1\) [34]. We then resized and reduced the complexity of the triangulated mesh using Autodesk Meshmixer\(^2\) and imported it directly into our optimization environment; the final mesh (Figure 8b) had 626 triangles. Figure 12 shows the initial condition of the optimization.

Figure 13 shows the optimized shape for the single point case. The drag decreased 61.3% compared to the grossly oversized baseline single point case. The optimizer generated a rounded leading edge and an elongated trailing cone with moderate closure angle. In this case, the rotational asymmetry is due to the asymmetric constraint geometry, not the flow condition.

Figure 14 shows the optimized shape for the multipoint case. Drag decreased 78.5% compared to the
grossly oversized baseline multipoint case. Compared to the single point case, we see an even blunter leading edge and dramatically shorter overall length. The result is an even larger thickness-to-chord ratio, which reduces drag in the crossflow condition. The multipoint case benefits more from shape optimization than the single point case. This may be due to in part to the better relative performance of the initial NACA 0012 geometry in oncoming flow compared to significant crossflows.

We then tightened the spatial tolerance and re-ran the multipoint case. Figure 15 shows that the spatial fit has tightened significantly compared to Figure 14. There is a noticeable decrease in frontal area of the fairing. Drag decreased an additional 16% compared to the $\rho = 1200$ case. This illustrates the direct design tradeoff between spatial requirements and aerodynamics.

5 Conclusions

Geometric (or spatial) integration constraints have been a barrier to industrial adoption of high-fidelity MDO for vehicle design applications. Existing geometric constraint techniques are limited in their ability to capture complex geometries. We reviewed the state-of-the-art in packing optimization problems and identified gaps that prevent existing methods from being directly applied to aerospace MDO. We then developed a mathematical approach for imposing spatial integration constraints based on triangulated surfaces and con-
constraint aggregation using the KS function. This allows practitioners to specify geometric design intent, even for complex geometries, in an intuitive and visual way. The method is general, differentiable, and efficient, making it suitable for use with gradient-based optimization with high-fidelity simulation tools.

We demonstrated the performance of our method by minimizing drag for three test geometries of increasing complexity. Drag decreased by 30% to 80% compared to the initial designs. Multipoint aerodynamic problems with crossflows benefited more from shape optimization generally. Tightening spatial tolerances reduced drag as expected. By varying the spatial tolerance $\rho$, designers can perform high-fidelity trade studies to quantify the value of tighter systems or payload packaging. All optimizations, including the multipoint 3D cases, were run on a desktop computer in less than 24 hours, placing the shape optimization capability in reach of typical MDO engineers. The new constraint formulation is a major improvement over point thickness geometric constraints and an important step towards solving aircraft systems packaging MDO problems.

Acknowledgments
The first author was supported by the National Science Foundation Graduate Research Fellowship Program under Grant DGE 1256260. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work was also supported in part by the U.S. Air Force Research Laboratory (AFRL) under the Michigan–AFRL Collaborative Center in Aerospace Vehicle Design (CCAVD), with Dr. Richard Snyder as the task Technical Monitor.

References


