12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSM
17 - 19 September 2012, Indianapolis, Indiana

AIAA 2012-5605

GeoMACH: Geometry-Centric MDAO of Aircraft Configurations with High Fidelity

John T. Hwang * and Joaquim R. R. A. Martins †
University of Michigan, Ann Arbor, Michigan, 48109, United States

This paper presents GeoMACH: Geometry-centric MDAO of Aircraft Configurations with High fidelity. GeoMACH is an open-source aircraft design tool suite under development that is planned to support MDAO with a large number of design variables. First, the overall GeoMACH architecture is described, including a proposed method for streamlined data transfer between disciplines. GeoMACH's efficient and lightweight B-spline engine is then introduced as it forms the basis for the geometry-centric approach to MDAO. Next, GeoMACH's OML and structural modelers are presented, which provide parametric aircraft modeling tools that span a configuration-level design space.

I. Introduction

COMPUTATIONAL tools have undoubtedly had a profound influence on the commercial aircraft design process, but their full potential is far from realized. In terms of impact, design methods such as multidisciplinary design optimization (MDO) have lagged behind analysis tools, such as computational fluid dynamics (CFD) and finite element analysis (FEA), both of which are now well-integrated into practical aircraft design. CFD and FEA solvers improve with every advance in algorithms and hardware, enabling a more accurate prediction of the 'perfect' design; yet, equally important are the design methods and processes that use these tools to find that 'perfect' design.

If a sufficiently robust and efficient MDO algorithm were possible, the pursuit of the 'perfect' aircraft would simply amount to solving for the mathematical optimum of the MDO problem with all disciplines, constraints, and design variables considered. While no such algorithm exists, many high-fidelity tools developed by the research community represent significant steps towards achieving this idealistic goal. For instance, Hicken and Zingg [1] developed a CFD-based aerodynamic shape optimization algorithm capable of optimizing various nonplanar wing designs. An encouraging result was the recovery of the winglet as a local optimum in induced drag minimization. Mader and Martins [2] presented a stability-constrained aerodynamic shape optimization algorithm that also uses CFD and applied it to shape optimization of a flying wing subject to stability, trim, and bending moment constraints, capturing the expected trade-off between washout and reflex airfoils for trimming the wing. An example of MDO is the work of Kenway et al. [3] involving aerostructural optimization with CFD and FEA of a thin-walled structure to maximize range with respect to shape and structural sizing variables. Convergence required a wall time of only 36 hours on 435 processors for MDO with respect to 476 design variables and 7 operating conditions on a 2 million-cell CFD mesh and 300,000 structural degrees of freedom (DOF). Kennedy and Martins [4] added composites and a panel-level design component to a similar aerostructural optimization problem, using a 3-D panel code instead of an Euler solver as was the case in the previous three examples. Optimization was performed with respect to 446 aerodynamic, structural, and material design variables subject to 871 constraints.

While these algorithms represent the current state of the art, two obstacles limit their impact on the design of commercial aircraft. First, each of these research efforts is highly specialized — they address just one aspect of the airframe design problem in great detail, though together they cover the aerodynamics, structures, stability, and materials disciplines. These algorithms are finely tuned to solve their respective problems extremely efficiently and reliably in order to push the envelope in number of operating conditions, design variables, and constraints, but this is sometimes done at the expense of versatility. For instance, they are not always designed to handle a wide range of geometries, to be user-friendly, or to allow seamless transitions between disciplines and between solvers, restricting their utility in practical aircraft design. However, their results and contributions are invaluable for more applied research programs. The second challenge is the difficulty of utilizing these algorithms in the absence of a 'push button' MDO algorithm that can consider all disciplines and variables simultaneously. Combining several disciplines would introduce local minima in addition to a substantial increase in the total number of state variables, design variables, objectives, and constraints, prohibiting a simultaneous, all-inclusive MDO. The objective is to address these two

*Ph.D. Candidate, Department of Aerospace Engineering, AIAA Student Member
†Associate Professor, Department of Aerospace Engineering, AIAA Senior Member

challenges through design tools that are more modular and a design method that can work within and overcome the limits of computing power and algorithms available today.

The proposed solution is GeoMACH: Geometry-centric MDAO of Aircraft Configurations with High fidelity. GeoMACH is intended to be an integrated aircraft design tool suite which streamlines interaction with the user through visualization and a simple interface. The rationale is to support high-fidelity multidisciplinary analysis and optimization (MDAO) with any combination of disciplines, variables, configurations, and solvers, and to overcome the lack of a 'push button' MDO solver using the creativity and knowledge of the human designer. A key enabling component is a central geometry modeler which facilitates communication among disciplines, supports unconventional configurations, and gives the user or optimizer simple but powerful parametric control of the outer mold line (OML) and structural model. The expected result is a unified design environment in which the user has access to a full array of tools including initial geometry creation, intuitive geometry manipulation, single- and multi-disciplinary analysis and optimization with a large number of design variables, and visualization of results.

The complete set of features for the envisioned aircraft design environment is not yet implemented; however, the core components are in a working state. This includes the B-spline engine, OML modeler, and structural modeler, all of which also have value as standalone tools. Thus, the components are in place for GeoMACH's primary capabilities: geometry creation and manipulation, generation of the structural model, and the ability to interface to external meshes and automatically morph them in response to changes in GeoMACH's parametric OML model.

This paper presents the MDAO approach, envisioned framework, and the developed OML and structural modelers. Particular emphasis is placed on aspects of GeoMACH which represent original contributions with value beyond the current implementation. First, an overview of the proposed tool suite is given, including details on architecture and the adopted approach to MDAO. Second, GeoMACH's B-spline engine is presented and described in depth as it provides a foundation for both the OML modeler and communication between disciplines. Third, GeoMACH's OML modeler is introduced and sample configurations produced by the OML modeler are shown. Next, the structural modeler is presented, including details on the parametrization, the underlying algorithms, and some sample structures. Finally, the contributions that have emerged during the development of GeoMACH are highlighted and summarized.

II. The GeoMACH Tool Suite

GeoMACH is meant to be an open-source aircraft design tool suite with an array of tools for MDAO. As such, development is ongoing, and it is anticipated that the list of components and the components themselves will evolve. However, there are several core features that are expected to be permanent, defining characteristics of GeoMACH. These are the subject of this section and are organized into the following subsections: implementation, approach, and architecture.

A. Implementation

GeoMACH is planned to be implemented in OpenMDAO [5], a Python-based open-source framework for multidisciplinary analysis and optimization developed at the NASA Glenn Research Center. The objectives of OpenMDAO are in line with the objectives of GeoMACH — a unified environment used to facilitate the integration of analysis tools from multiple disciplines to solve engineering design problems using a combination of analysis, optimization, MDA, and MDO. By building on top of the OpenMDAO framework, GeoMACH leverages existing capabilities and adds new tools specific to conceptual and preliminary design of aircraft.

The use of OpenMDAO yields four benefits that address some of the requirements for GeoMACH. First, any use of OpenMDAO is inherently modular because of the infrastructure of OpenMDAO. Problems are decomposed into components, which are Python objects that accept arguments, perform computations, and return some result. Links between the inputs and outputs of different components can be specified, and OpenMDAO handles the data passing. Second, OpenMDAO has a library of wrapped optimizers and MDO architectures that can be automatically implemented. Third, GeoMACH can take advantage of OpenMDAO's graphical user interface and visualization capabilities which are currently being developed within a browser-based environment. Finally, automatic and flexible methods for computing derivatives are under development, which will eventually be added to OpenMDAO's set of features. Recently, it has been shown that the known methods for computing derivatives, including adjoint-based methods, can be unified under a common framework [6], which would make it possible to modularize derivative computation further and handle most of the effort at the framework level.

As with OpenMDAO, the primary programming language that GeoMACH will use is Python. Python is an easy-to-use scripting language that can wrap codes written in compiled languages such as Fortran, C, or C++. Therefore, the approach used in GeoMACH is to implement most components in high-performance compiled languages and to use Python to integrate them, taking advantage of object-oriented programming at the Python level.

B. Approach

The basic components that are required for MDAO include an optimization algorithm, geometry engine, MDO architecture, aerodynamics solver, structural solver, and other solvers for each discipline of interest. With gradient-based optimization, sensitivity analysis, or Newton-type multidisciplinary solvers, a module for efficient gradient computation is an additional necessary component. For each of these components, there are many available options corresponding to different computational costs, complexities, approaches, and levels of accuracy, but it is not clear which set of choices is the most prudent. However, these choices are often guided by the geometry engine that is selected due to the central role that it often plays in an aircraft design framework.

GeoMACH is aptly described as taking an OML-driven approach since the OML is parametrically represented with a large amount of detail and all of the disciplines derive their respective models or meshes from this central model. There are many aircraft design frameworks and geometry engines in the literature which adopt a similar philosophy. This includes VSP (Vehicle Sketch Pad) [7], MICADO [8], RAGE (Rapid Geometry Engine) [9], and AVID PAGE (Parametric Aircraft Geometry Engine) [10].

In addition to the fact that they are parametric, all of these existing geometry engines also share a similar overall approach. In each case, aircraft components such as the wing and fuselage are individually created, typically by lofting curves, and the intersections between components are computed as a necessary step in achieving a closed surface. Often, the underlying geometry for each component is represented as parametric surfaces such as non-uniform rational B-spline (NURBS) surfaces. To produce a discrete surface representation, triangulation is required at this point as the intersections are parametric curves which delimit surface regions that are no longer exposed. In all cases, the final tessellation is in general an unstructured mesh. For high-fidelity aerodynamic or structural analysis, the next step is to create a CFD mesh or structural mesh, respectively. With VSP for instance, there are tools for automatic generation of CFD volume [11] and structural [12] meshes. If a structured multi-block CFD solver is to be used, mesh generation is necessarily a manual process but is still possible given a geometry represented by trimmed surfaces.

The aforementioned geometry engines are useful tools that facilitate analyses and small-scale optimizations at the conceptual design level; however, they reach their limits with optimization involving many design variables. GeoMACH is intended to be a high-fidelity MDAO tool suite supporting single- and multi-disciplinary optimization with a large number of design variables, as it is the view of the authors that this is the most promising avenue for research with the greatest potential for making a significant impact on the design of commercial aircraft.

All of the examples of optimization discussed in Section I involve hundreds of design variables and expensive CFD and FEA analyses with hundreds of thousands to millions of state variables. Convergence of these optimizations in a reasonable amount of time is only possible with the orders-of-magnitude reduction in computation time provided by sound MDO architectures [13], gradient-based optimization [14], and adjoint-based computation of derivatives [6]. Geometry engines that rely on intersection to produce closed surfaces are not compatible with this approach due to several reasons. First, the lack of smoothness in the geometry parametrization can cause noisy gradients and difficulties in converging the optimization problem. Moreover, the intersections must be recomputed in each optimization iteration, and the mesh must be regenerated as mesh movement algorithms fail in the presence of topological changes in the surface mesh. With mesh regeneration, the only feasible approach is to use a gradient-free optimizer, which makes the computational cost prohibitive for the types of problems discussed in Section I, even with an embarassingly parallel implementation.

The algorithms listed in Section I do not make use of conceptual design geometry engines. Their parametrizations involve direct manipulation of the control points of the B-spline surfaces [1] or free-form deformation (FFD) blocks which morph a discrete model imported from an external source [3]. In each case, the restricted parametrizations enable high-fidelity optimization with many design variables; however, these parametrizations do not span the full conceptual design space.

The survey of the literature suggests that there is a need for a configuration-level aircraft geometry engine that supports high-fidelity MDAO with a large number of design variables. Existing aircraft geometry engines succeed in capturing the full breadth of configurations but are not compatible with the MDAO problems of interest. Existing geometry parametrizations (Section I) that support MDAO with large numbers of design variables are not sufficiently versatile. Further, there are additional requirements such as a smooth parametrization of the surface mesh and support for structural multi-block CFD. The latter requirement derives in part from the fact that the most efficient mesh movement algorithms apply the linear elasticity equations on a coarsened version of a structured mesh.

GeoMACH's approach is to model the aircraft as a union of untrimmed surfaces — i.e. surfaces with four topological edges. Adopting a structured representation enables simple parametrizations, similar to those of the components in the surveyed geometry engines. Furthermore, the need to compute intersections can be avoided through the use of interpolating surfaces that enforce the desired level of continuity. More details are given in Section IV.

C. Architecture

The high-fidelity optimization algorithms discussed in Section I cover the aerodynamics, structures, materials, and controls disciplines. In conceptual design and the early parts of preliminary design, these are arguably the driving disciplines that must be considered at this level because of their strong coupling to each other. The weights discipline is another driving factor in the design of the airframe, and propulsion-airframe coupling also becomes significant when comparing configurations.

Based on the requirement of supporting high-fidelity MDAO with many variables, GeoMACH makes a number of assumptions about the types of analysis tools that will be used. As mentioned previously, it is assumed that the CFD solver uses structured multi-block grids that are driven by mesh movement algorithms that combine the algebraic and linear elasticity-based approaches [15, 16]. The structural solver must handle shell elements as GeoMACH models the aircraft as a thin-walled structure. A materials component is planned to be integrated, with a library of material types including laminated, textile, and sandwich composites. The approach is to develop and implement micromechanical models that parametrically yield the shell stiffness matrices in terms of the material design variables. An example of such a model derived through homogenization for a textile composite is given in [17]. For propulsion-airframe coupling, an inexpensive approach is to use a 0-D model for the engine and parametrize the boundary conditions on the inlet and outlet surfaces of the CFD grid based on the output from this model. For controls, a first approximation is to enforce static stability using a finite-difference approximation for the derivative of the moment coefficient with respect to angle of attack. In each discipline, alternate approaches are possible as long as their outputs vary smoothly and they are compatible with high-fidelity MDAO. Figure 1 shows that each discipline interacts with the aircraft geometry through either the OML or structural model.

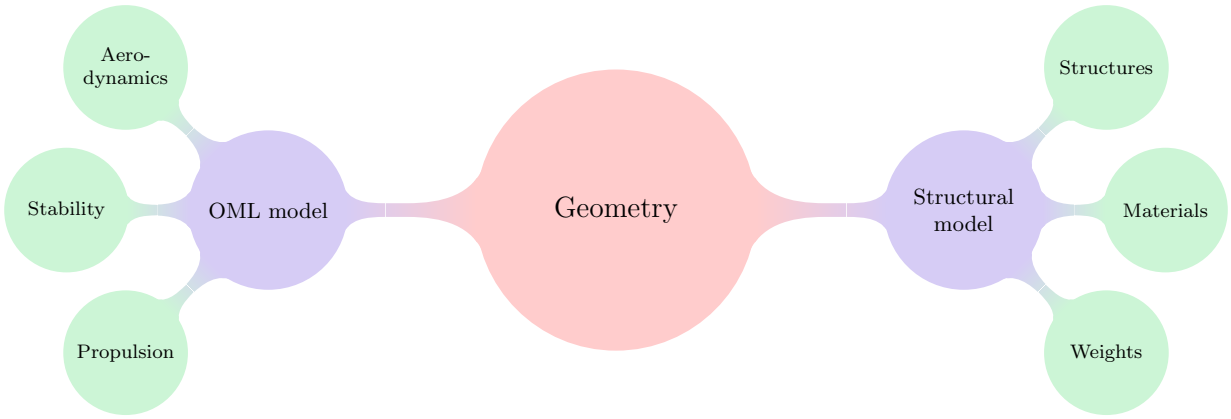


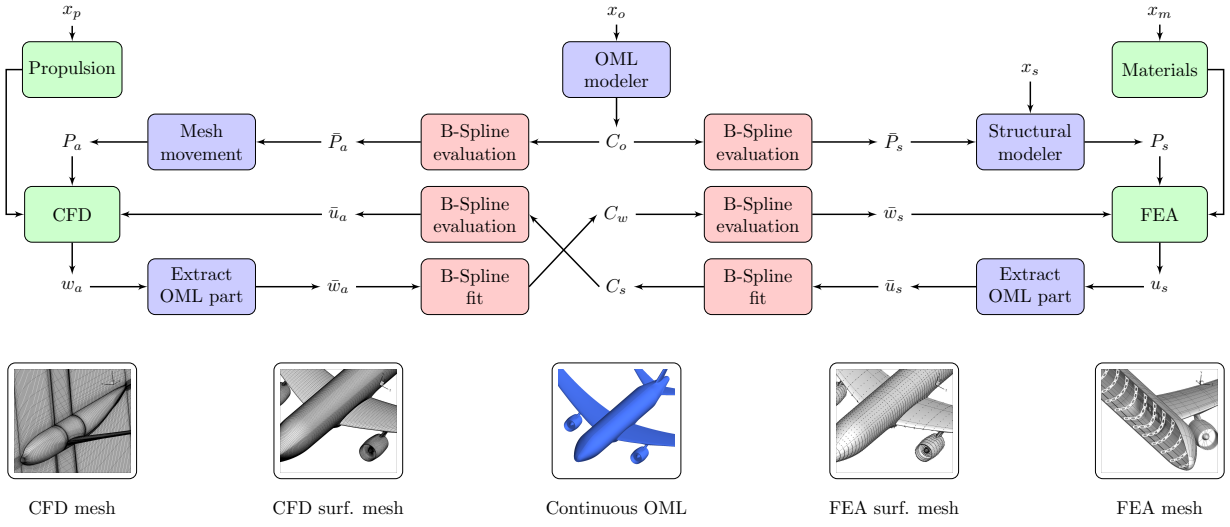
Figure 1: Illustration of how each discipline interacts with the geometry. The aerodynamic mesh is linked to the OML model through the mesh movement algorithm. Stability coefficients and their derivatives are computed by the flow solver, and the quantities computed by the propulsion model determine the inlet and outlet CFD boundary conditions. The materials discipline provides stiffness matrices for the shell elements used in the structural analysis, and a significant portion of the weight build-up depends on the airframe.

GeoMACH's efficient B-spline engine contains much of the machinery which handles interaction between disciplines and data transfers between disparate models. The B-spline engine is described later, in Section III. However, the architecture and organization of components are discussed here and are shown in Figure 2.

Since B-splines are linear in the control points, they allow simple and efficient forward and inverse evaluation. In Figure 2, the B-spline engine plays two important roles. First, they provide a continuous, discipline-independent representation of the OML. The parametric OML modeler accepts shape design variables as input and computes the appropriate values for the OML control points to achieve the desired shape. The shape design variables can be as high-level as span or sweep, or as fine as the perturbation of an individual control point. Next, two pre-computed matrices map the OML control points to the vectors of CFD and FEA mesh points that lie on the OML. In the case of CFD, the B-spline Jacobian matrix can be computed by extracting the OML points of a manually created CFD mesh and passing it to GeoMACH's B-spline engine. It would then perform inverse evaluations to find the parametric coordinates of each point and evaluate the B-spline basis vectors for each point, which combine to form the Jacobian. The only requirement is that the surfaces of the manually created CFD mesh match with the OML. A simple matrix-vector product between the computed Jacobian and the OML control points yields the new CFD surface mesh points

reflecting the changes in the shape design variables. The mesh movement algorithm then takes the vector of CFD surface mesh points and warps the rest of the mesh to attempt to preserve mesh quality. On the structures side, the process is similar except the mesh movement algorithm is replaced by the structural modeler which takes the surface points of the structural mesh and generates the geometry of the internal structure. Details on this are given in Section V.

The second important role of the B-spline engine is load and displacement transfer. The aerodynamic states computed by the CFD solver must be converted to a form in which it can be applied as loads on the FEA mesh, and structural states computed by the FEA solver must be converted to displacements that can be applied to the CFD mesh, though the two meshes are not coincident. The two B-spline Jacobians can be re-used to perform a least-squares fit and find the values of the 'control points' for each discipline's state variables, which essentially amounts to interpolation of the state variables on the B-spline basis. Once the control points of the interpolants are computed, the two sets of states variables 'cross over' — the vector of control points for the structural states is left-multiplied by the Jacobian for the aerodynamic mesh and vise versa. The cost of computing the fit can be mitigated by performing an initial direct factorization of $B^T B$ for the two disciplines, where B is the B-spline Jacobian. This step is relatively inexpensive since the size of the system is the number of control points, which is typically on the order of ten times smaller than the number of OML points. It remains to be seen whether this proposed technique for load and displacement transfer is sufficiently accurate and mechanically sound; if so, it would present an interesting alternative to load and displacement transfer using rigid links [18], which is commonly used in high-fidelity MDO.



CFD mesh | CFD surf. mesh | Continuous OML | FEA surf. mesh | FEA mesh

Quantities on the CFD mesh	
P_a	CFD mesh coordinates
$\bar{P}_a$	Subset of CFD mesh points that are on the OML
w_a	Aerodynamic states on the CFD mesh
$\bar{w}_a$	Aerodynamic states on the OML part of the CFD mesh
$\bar{u}_a$	Structural states on the OML part of the CFD mesh

Neutral quantities	
x_p	Propulsion DVs
x_o	OML (shape) DVs
x_s	Structural DVs
x_m	Material DVs
C_o	B-spline control points for the OML
C_w	Control points for the interpolated aero. states
C_s	Control points for the interpolated str. states

Quantities on the FEA mesh	
P_s	FEA mesh coordinates
$\bar{P}_s$	Subset of FEA mesh points that are on the OML
u_s	Structural states on the FEA mesh
$\bar{u}_s$	Structural states on the OML part of the FEA mesh
$\bar{w}_s$	Aerodynamic states on the OML part of the CFD mesh

Figure 2: Data flow diagram showing GeoMACH's proposed architecture. Each variable shown is a vector of quantities, where x represents design variables, P represents discrete surface or mesh points, C represents B-spline control points, u represents structural states (displacement and rotations), and w represents aerodynamic states (pressures, momentums, etc.). The subscript p represents propulsion, o represents OML, m represents materials, a represents aerodynamics, and s represents structures.

III. B-Spline Engine

A core aspect of GeoMACH's approach to MDAO is the use of B-splines for representing geometry through an efficient, lightweight B-spline engine. A B-spline is a piecewise polynomial function whose degree and smoothness can be specified. B-spline curves represent the generalization of Bezier curves in a manner similar to the way splines generalize Lagrange interpolants — using piecewise functions to allow a lower degree. As such, they smoothly approximate a set of control points, which suggests their utility in fitting discrete curves, surfaces, etc., with fewer degrees of freedom. A useful property is that B-splines are splines of minimal support [19]; therefore, each row of the Jacobian of evaluated points with respect to control points has a small, fixed number of non-zero entries. B-spline surfaces, volumes, etc. can be formed by multiplying together B-spline functions in each parametric direction — these are known as tensor-product B-splines.

There are many reasons why B-splines are well-suited for geometry modeling, but four are particularly relevant here. First, they are accurate because they have analytical definitions that allow them to easily support perfectly watertight geometries through exact union of patches. Second, B-splines have direct, analytical definitions, and the fact that they are piecewise polynomials is attractive because their evaluation involves only arithmetic operations as opposed to elementary functions such as sine and cosine. Third, the derivatives of B-splines with respect to parameters are analytically defined, and they are linear in the control points, yielding both accuracy and efficiency in computing derivatives. Finally, the fact that a B-spline has a small support means that the Jacobian of evaluated points with respect to the control points is sparse.

GeoMACH's B-spline engine treats the entire patchwork of B-spline surfaces as a single continuous entity. It stores and works with a unique list of points and control points as shared edges and vertices point to the same elements. Figure 3 illustrates how this guarantees that the model stays watertight. In GeoMACH, the B-spline surfaces use open uniform knot vectors, which means the edge of a tensor-product surface is exactly equal to the curve generated by the edge control points and that the four corners or vertices of the surface are exactly the control points at the corners. This is built into the B-spline Jacobian matrix, eliminating the necessity to manually enforce continuity along shared edges and vertices. GeoMACH's B-spline engine directly assembles and stores the Jacobian matrix, so an evaluation of the B-splines reduces to an inexpensive matrix vector product.

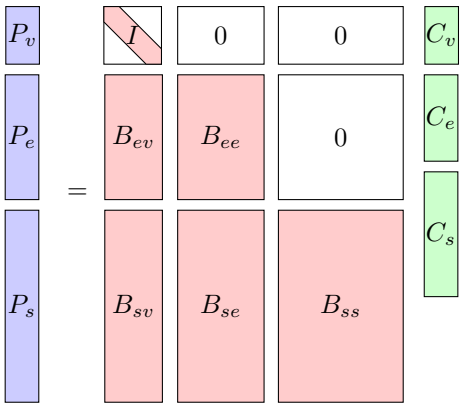


Figure 3: A block matrix diagram illustrating how GeoMACH's B-spline engine evaluates the patchwork of surfaces, ensuring that it is watertight. P represents the vector of evaluated points, C represents the vector of B-spline control points, and B represents the B-spline Jacobian. The subscript v stands for vertex, e stands for edge, and s stands for surface.

GeoMACH's B-spline engine is always initialized by importing a list of 2-D arrays of points. It then computes the connectivities of vertices and edges that are coincident to within a tolerance, initializes knot and parameter vectors, and assembles the Jacobian. Some timings are given in Table 1 for the most computationally intensive steps for some of the sample aircraft configurations that are shown in Section IV.

The projection algorithm is of particular interest because it is used in linking the CFD and structural meshes to the B-spline OML model. More specifically, each of the CFD and structural mesh points that lies on the OML is projected onto the B-spline model to determine the parametric coordinates corresponding to the point. The algorithm uses as the

		Conventional	Strut-braced	D8	Joined wing	Sphere
Count	# Surfaces	165	160	98	142	5
	# Control points	6244	5082	2592	6027	18600
	# Points	63713	51846	25001	62662	2994004
	# Jacobian non-zeros	938281	756204	359848	927066	47784232
Times (s)	Connectivities	0.0243	0.0229	0.0107	0.0187	0.00044
	Jacobian assembly	0.107	0.0826	0.0378	0.0101	5.05
	Fit control points	0.166	0.146	0.0762	0.177	5.7
	Evaluate points	0.00847	0.00748	0.00349	0.00841	0.23

Table 1: B-spline engine timings for four aircraft configurations and a fine representation of a sphere.

initial guess the closest point on the discrete B-spline model and applies Newton's method with analytic derivatives. The algorithm detects when the update takes the one of the parameters outside the knot span and applies appropriate convergence checks to terminate if the solution is outside the knot span. Sample timings are given in Table 2 for structural mesh generation for the conventional configuration.

# Surfaces	# Points	# Total time (s)	# Time/projection (s)
51	9336	38.26	8.04×10^{-5}
	20781	85.24	8.04×10^{-5}
16	4032	1.56	2.42×10^{-5}
	9072	3.56	2.45×10^{-5}
10	3156	0.87	2.76×10^{-5}
	7002	1.97	2.81×10^{-5}
1	1512	0.128	8.47×10^{-5}
	3402	0.288	8.47×10^{-5}

Table 2: B-spline engine projection timings. The first column represents the number of surfaces that must be checked.

The B-spline engine has multiple options for exporting geometries. One option is to export to an IGES file, a computer-aided design (CAD) format, which preserves the continuous representation of the geometry by storing the knot vectors and control points. Among discrete alternatives, it can produce an unstructured surface mesh by splitting every quad into two triangles and exporting a unique set of vertices and a list of triangles defined by triplets of these vertices. An option that preserves the structured format is to output a list of surfaces; however, connectivity information at the surface level is lost in this format. If no connectivity information is necessary, it can also output a list of unique points, which is its native format.

IV. OML Modeler

GeoMACH's OML modeler is responsible for both the creation and parametrization of the aircraft's OML and uses the B-spline engine to represent the OML. There are three requirements for the OML modeler that derive from those of the overall GeoMACH tool suite. The first requirement is that the parametrization for the OML modeler is highly automated. This flows down from the top-level requirement that all components must be easy to use and have streamlined user interaction so that geometry manipulation, analysis, and optimization are a routine process. In terms of the OML modeler, this translates to the ability to specify a small number of high-level aircraft design parameters if desired — such as span, sweep, linear twist, and linear taper — and obtain useful designs without much manual effort. It is critical that this is both intuitive and simple for GeoMACH to be useful as an aircraft conceptual design tool. The second requirement is that the parametrization provides sufficient depth and detail. Since GeoMACH is designed to support high-fidelity shape optimization with many variables, the OML modeler must be able to provide precise control for the optimizer to maximize its capabilities. It is also necessary for the OML modeler to be robust and versatile enough that the user has multiple options for controlling a shape variable at various degrees of precision. For instance, the user should be able to choose a single value for the chord of a wing, a quadratic taper along the span, or chords at multiple span-wise stations which give access to a large range of smooth planform shapes. The final requirement is that GeoMACH is versatile enough to handle a conceptual design space. This includes conventional and unconventional configurations with all of the main lifting surfaces and bodies modeled, as well as support for

configuration-level design changes such as control over the location of the wing relative to the fuselage. The following sections describe the OML modeler in detail and show how these requirements are satisfied.

A. Architecture

The requirements for the OML modeler are inherently conflicting, which presents a challenge for its design. It is desired that the OML modeler is fully parametric and as automated as possible; however, with more precision and depth of control, the user must inevitably specify more information. Furthermore, a design space containing multiple configurations yields a wide variety of parametrizations, and the automation and detail must be maintained.

The OML modeler provides a solution that satisfies these conflicting requirements by conceding that a configuration-level design space cannot be spanned continuously. The premise is that there is a finite number of configurations of interest, and since GeoMACH is open-source, the plan is to have a library of configurations which all users can contribute to and pull from. This library would also contain structured multi-block CFD meshes corresponding to each configuration, which, combined with a robust and efficient mesh movement algorithm, allows the library to span a full configuration-level design space. Thus, improving the reliability and capabilities of mesh movement algorithms is a critical direction for future research; however, there has been recent ongoing progress in this area. Figure 4 shows a CFD solution for the double bubble aircraft configuration.

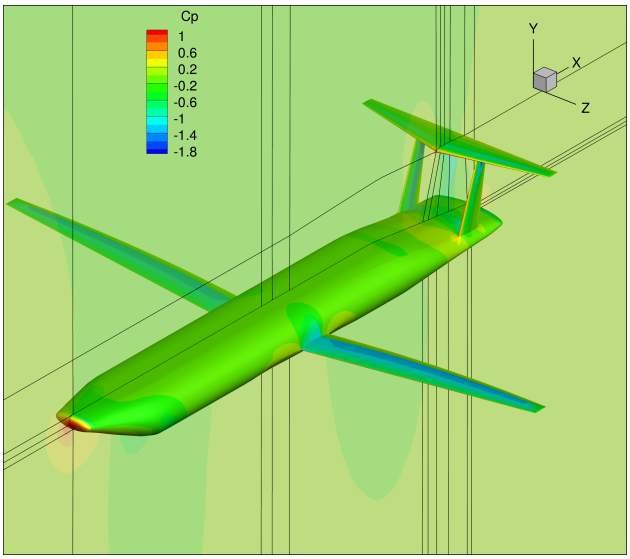


Figure 4: CFD solution for the double bubble aircraft configuration at $M = 0.78$ and $\alpha = 2°$.

The creation of a configuration is greatly simplified by the fact that they are built by assembling components from one of three basic types — the body-type, wing-type, and junction-type. As a result, full aircraft configurations, such as those shown in Figure 5, can be created with on the order of only tens of lines of code, as shown below:

```
self.addComp('fuse', Body([70,10,10,20,10,10,10,50,10,25,10,10],[25,25,25,25],[15]))
self.addComp('wing', Wing([10,10,10,50],[10,20,10,10]))
self.addComp('tail', Wing([30],[25]))
self.addComp('nacelle', Body([50,10,20,30],[30],[20,10,10,20],full=True))
self.addComp('pylon', Wing([30],[20],opentip=True))
self.addComp('fin', Wing([30],[25],half=True))

self.separateComps()

self.addComp('wingfuse', FullInterface(self.comps, 'wing', 0, 'fuse', 2, [2,1], [3,6]))
self.addComp('tailfuse', FullInterface(self.comps, 'tail', 0, 'fuse', 2, [1,8], [2,10]))
self.addComp('pylonnacelle', FullInterface(self.comps, 'pylon', 0, 'nacelle', 1, [1,1], [2,3]))
self.addComp('pylonwing', FullInterface(self.comps, 'pylon', 1, 'wing', 1, [2,1], [0,2]))
self.addComp('finfuse', HalfInterface(self.comps, 'fin', 0, 'fuse', 1, [0,8], [0,10]))

self.assembleComponents()
```

This aspect of the OML modeler's design is what allows it to satisfy the requirements described in IV, as separating the conceptual design space into discrete configurations allows the full space to be spanned using only three types of components. Each component can then be given a robust and versatile parametrization that is detailed and allows control with only a few high-level parameters if so desired. Figure 6 shows how the influence of each component's shape design variables combines with the others to parametrize a common OML model.

The other function of the components is to generate the initial surfaces to load into the B-spline engine, in order to specify the connectivities between surfaces. Each component provides a list of surfaces represented as 2-D arrays of points for a simplified representation of the component — for a wing-type component this would be a thin plate, and a cube for a body-type component. Beyond the correct topology, the actual coordinates of the points are arbitrary since they are to be recomputed based on the aircraft design parameters. Once the full set of surfaces are imported into the B-spline engine, each surface is fitted with a corresponding B-spline surface whose control points are manipulated by the component classes through the appropriate parametrizations.

B. Wing Class

The Wing class is represented as two sets of surfaces — the upper and lower faces. C^1 continuity is enforced along the leading edge as well as the wing tip for instances of the Wing class that are not attached to another component at one end. The Wing class parametrization is simplified by dividing its control points into sections. Each section has an associated chord, x-y-z coordinates, and p-q-r rotations that can be set, but the Wing class also offers the option of automatically adding appropriate rotations to keep the sections normal to the tangent vector in the span-wise direction. This allows the user to choose rotated sweep instead of sheared sweep, for instance, and it is necessary for winglets, joined wings, and other non-planar wing designs. Euler angles are used, namely the x-y-z rotations — this applies rotations for dihedral first, then sweep, then twist. It is natural that twist would be applied last, but the x rotation is applied first because if the y rotation were instead, a wing with 45° rotated sweep could have a winglet at a 45° angle relative to the flow. Each section also has normalized x-y coordinates for its control points which are independent of the chord, translations, and rotations of the section, so these coordinates would give the normalized section airfoil. The Wing class has the ability to compute and perform a fit to any desired NACA airfoil or an airfoil given in a data file. This function uses the B-spline engine to interpolate the custom airfoil and fit a section of the OML model to it, so there are no restrictions on the format, number of points, or distribution of points used in the airfoil data file. As with the Body and Junction classes, the wing parametrization is computed at the Fortran level, but the subroutines are wrapped in a Python class.

The parametrization is further simplified by optionally parametrizing the section properties themselves with the use of piecewise polynomial functions. For sweep for instance, the user can specify a list of normalized span-wise coordinates from 0 to 1, a list of sweep values at each coordinate, and an optional argument for setting the order of each piecewise interpolant. The corresponding control points are computed, and since there is another layer of parametrization that maps the control points to the discrete points, the result is a smooth final geometry. This approach yields a clean and simple yet versatile method of parametrization which allows the user to provide a minimal amount of information. The following is a code excerpt from the conventional configuration in Figure 5 that demonstrates the simplicity of the parametrization.

```
c['wing'].offset[:] = [3.75, 0.3, 0.5]
c['wing'].setAirfoil("rae2822.dat")
c['wing'].props['posx'].set([0,3.2,4],[0,0.8,1],w=[0.4,1,0])
c['wing'].props['posy'].set([0,0.9,2.1],[0,0.8,1],w=[0.5,1,0])
c['wing'].props['posz'].set([0,4.5,5],[0,0.8,1],w=[0,1,0])
c['wing'].props['prpx'].set([1,1],[0,1])
c['wing'].props['prpy'].set([0,0],[0,1])
c['wing'].props['chord'].set([2,0.25],[0,1])
```

C. Body Class

The Body class is represented as five or six sets of surfaces, depending on whether it intersects the symmetry plane. Topologically, it is equivalent to a hexahedron with the front, rear, upper, lower, and side faces corresponding to the equivalent surfaces of a fuselage or engine nacelle. In a manner similar to the Wing class, the Body class makes use of sections of control points to simplify the parametrization. A section in the Wing class includes one column of control points from the upper and lower faces; a section in the Body class includes one column of control points from the

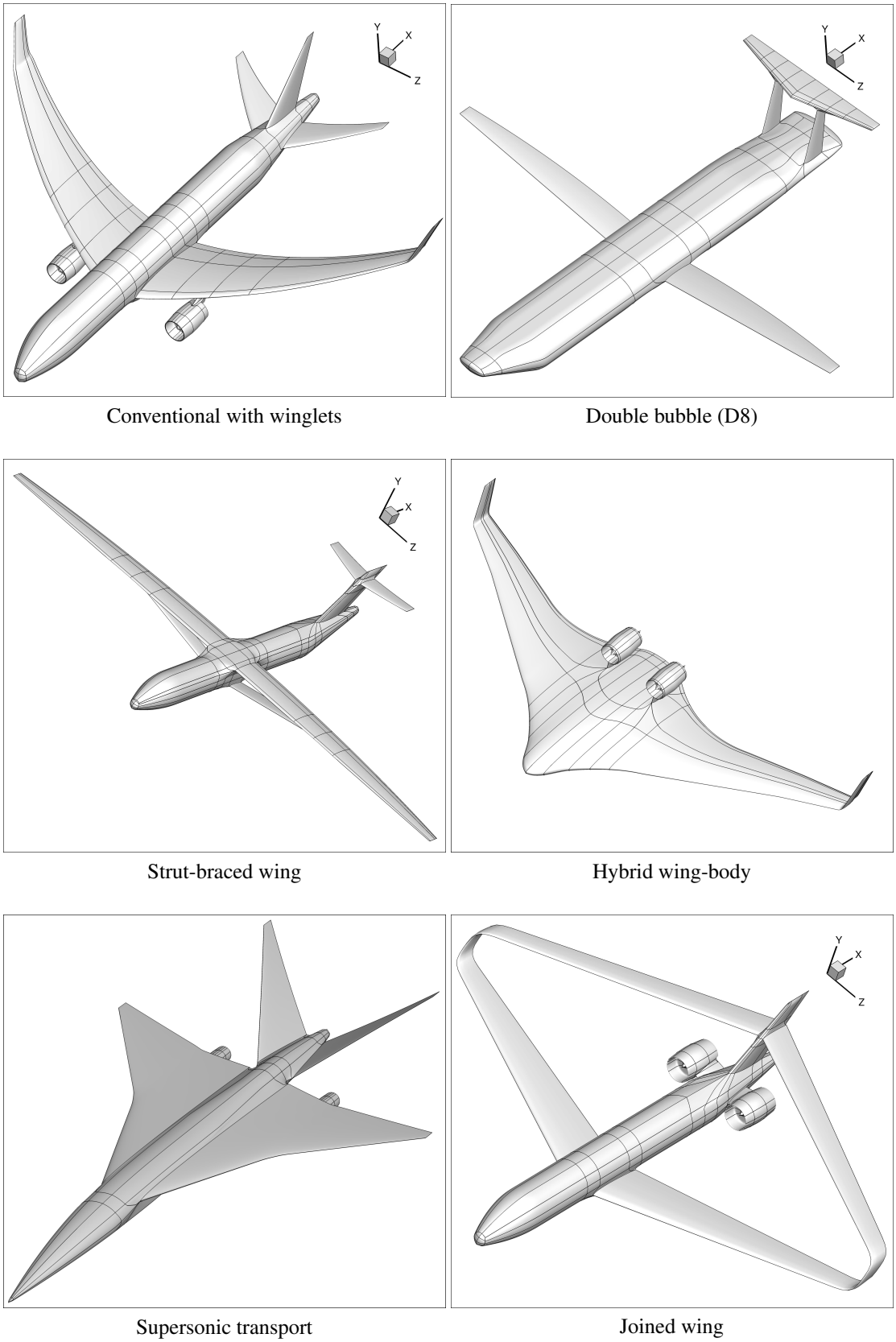


Figure 5: Aircraft models of six different configurations produced using GeoMACH's OML modeler. Each one is defined by only tens of lines of code.

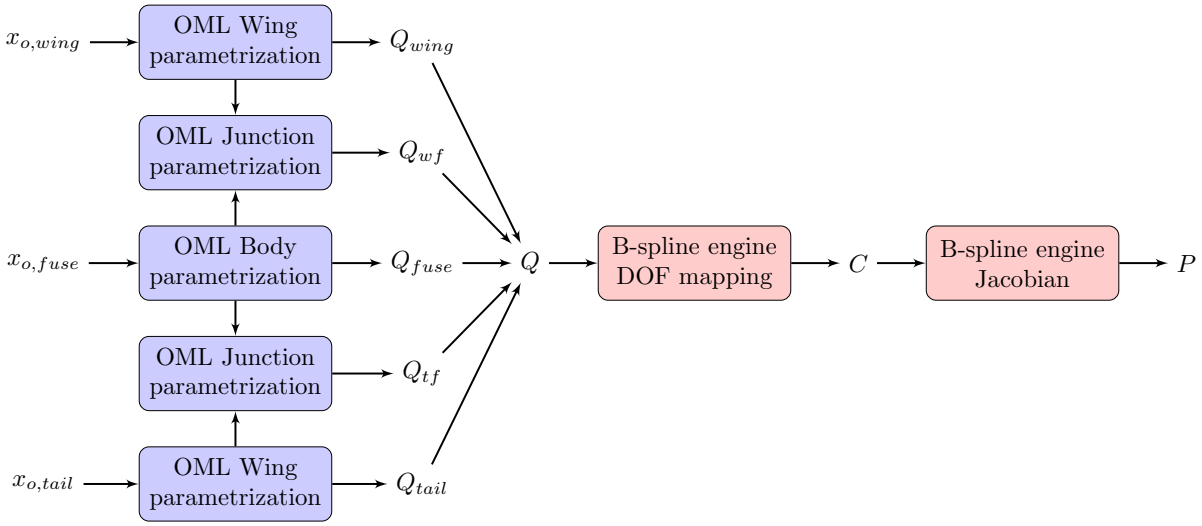


Figure 6: Illustration of the sequence by which shape design variables parametrize the discrete OML. The diagram corresponds to a wing-body-tail configuration, where the wing and tail are instances of the OML modeler's Wing class, the fuselage is an instance of the Body class, and the remaining components are instances of the Junction class. The variables are all vectors of quantities, where x represents design variables or parameters, P represents discrete OML points, C represents B-spline control points, and Q represents degrees of freedom — the set of control points with those along C^1 edges and vertices removed.

upper, lower, and side faces. Each section has x and y positions and radii in the y and z directions, and simple sections have been implemented including ellipses and rectangles with rounded corners. The latter can be used on the bottom half of the fuselage to emulate wing fairings. If desired, the section control points can be manually specified as well.

As with the wing, the section properties can be further parametrized with piecewise polynomials. Below is a code excerpt for the fuselage of the conventional aircraft in Figure 5.

```
c['fuse'].setSections(sections=[2,3,4,5], t1L=0.35, t2L=0.65)
c['fuse'].props['posx'].set([0,10],[0,1])
c['fuse'].props['posy'].set([0.3,0.5,0.5],[0,0.15,1],w=[1.0,0,0],d=[1,0,0])
c['fuse'].props['ry'].set([0.1,0.5,0.5,0.1],[0,0.15,0.75,1.0],w=[0.9985,0,0,0],d=[1,0,0,0])
c['fuse'].props['rz'].set([0.1,0.5,0.5,0.1],[0,0.15,0.75,1.0],w=[0.9985,0,0,0],d=[1,0,0,0])
```

D. Junction Class

The Junction class is what enables GeoMACH's OML modeler to produce watertight geometries with untrimmed 4-sided patches without computing intersections. It in turn is enabled by the simple, structured representations used by the Wing and Body classes. The surfaces of a sample Junction component are shown in Figure 7. Upon instantiation, a Junction object removes a set of surfaces from the component to which the attachment is made, which is the fuselage in the case of Figure 7. Next, the junction surfaces are added through linear interpolation. The surfaces in red are first computed by interpolating between the upper or lower part of the root airfoil section and the directly opposite horizontal edge above or below on the fuselage. The surfaces in blue are then interpolated from the left or right edges of the red surfaces and the far left and right edges from the fuselage. The computation of the blue surfaces effectively amounts to bilinear interpolation, unlike the red surfaces. Note that outside of the colored surfaces, the fuselage control points are unaffected by the junction and follow their original, circular shape.

The implemented junction surfaces work well, but they have two limitations. First, they only work for wing-type components that attach to the side of a body-type or wing-type component, so they cannot handle embedded engines such as those on the upper surface of certain blended wing-body designs. However, work is underway to improve the OML modeler so that embedded engines can be handled. Second, they work well when a non-smooth intersection is

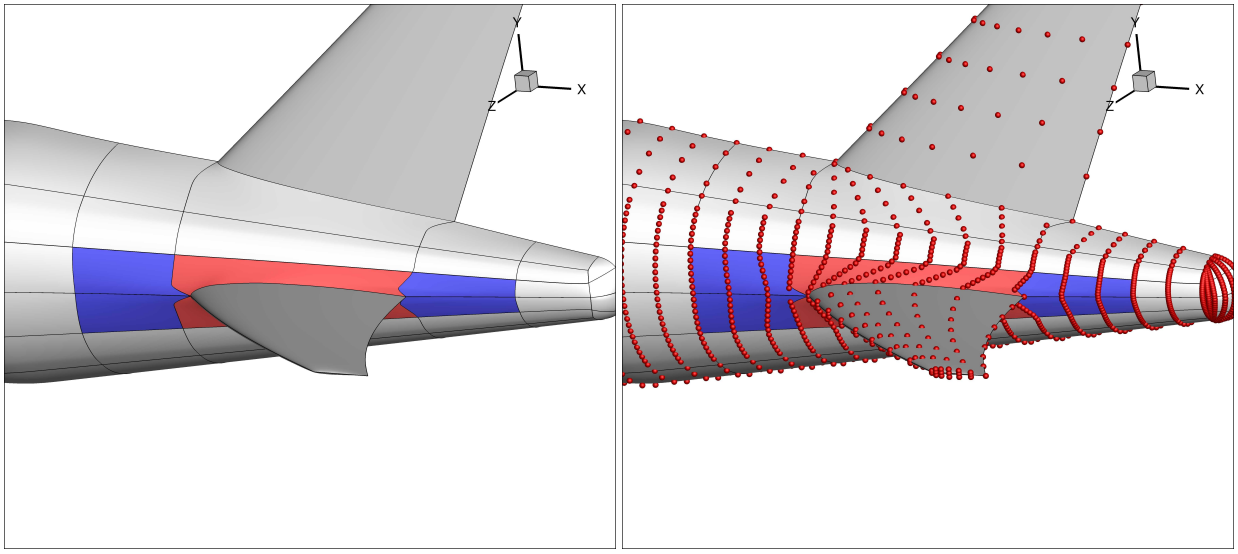


Figure 7: The tail-fuselage junction in a conventional aircraft. The colored surfaces represent the junction component and the small red circles are control points.

desired such as the tail-fuselage junction; however, C^1 may sometimes be desired to create a smooth transition from one component to the other.

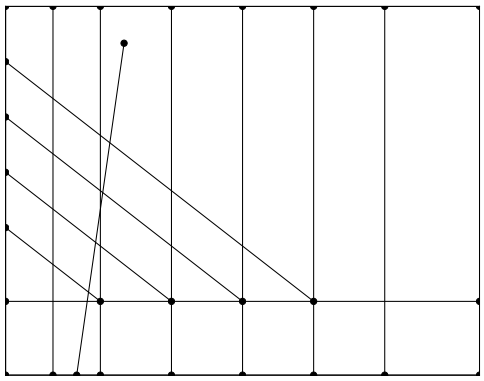
One proposed solution to the second limitation is to use a combination of Bezier curves and transfinite interpolation. In the case of the junction in Figure 7, this approach would first compute six quadratic or cubic Bezier curves: three leading up, left, and down from the leading edge of the root airfoil section and three leading up, right, and down from the trailing edge. Depending on the degree of the curve, a combination of C^0, C^1, or G^1 constraints can be used to compute the control points of the Bezier curve or additional degrees of freedom could be left as shape design variables for the junction. Using this net of boundary curves, transfinite interpolation can be used to compute all of the junction's B-spline control points. A similar approach is already used to compute the front and rear faces of the Body class. This approach could in fact be applied directly to the discrete surface mesh, bypassing the B-spline control points; however, either a fit would have to be performed after every shape variable update or the continuous representation of the OML would be lost.

V. Structural Modeler

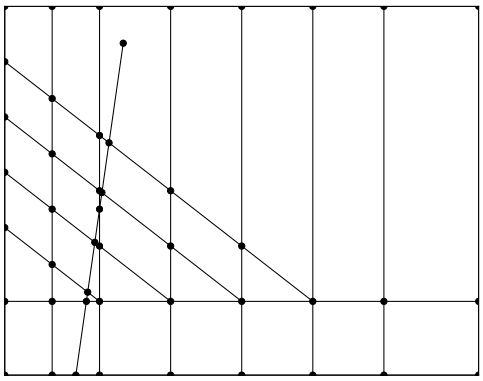
The role of the structural modeler is best illustrated by Figure 2. Based on the specified structural layout, it computes the discretization of the skin which is projected onto the surfaces of the B-spline model, and it also assembles the Jacobian of the full structural mesh with respect to the nodes on the skin. The matrix computed by the B-spline engine maps the OML control points to the structural surface mesh, and the matrix computed by the structural modeler maps the structural surface mesh to the full structural mesh. The two sections that follow describe the computation of the skin layout and the generation of the internal structure.

A. Skin Layout Generation

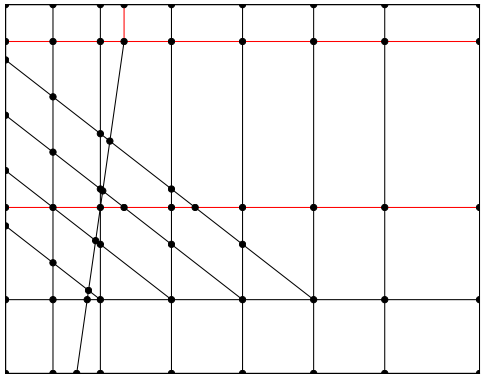
The skin for each component must contain edges at each location where an internal structural member or junction intersects the skin. Further, the domain must be a tessellation of quadrilaterals so that it can be populated with shell elements. Figure 8 shows how this tessellation can be computed from an arbitrary distribution of edges, given in 8(a). The first step is to compute all intersections between edges and create vertices at the intersection points, yielding 8(b). Vertices that are floating in space must be dealt with to be able to identify polygons in a meaningful way. The approach used to generate 8(c) is to loop through the vertices and add edges in the $0°$, $90°$, $180°$, and $270°$ directions, if there are not already edges in the $[-45°,45°]$, $[45°,135°]$, $[135°,225°]$, and $[225°,315°]$ quadrants, respectively. This process guarantees that the domain is covered fully by convex polygons with no more than five sides. Next, pentagons are split to obtain 8(d), and triangles are each split into three quadrilaterals to obtain 8(e). This final layout has a relatively large



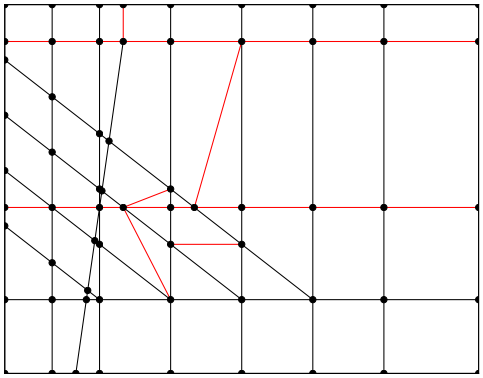
(a) Initial layout



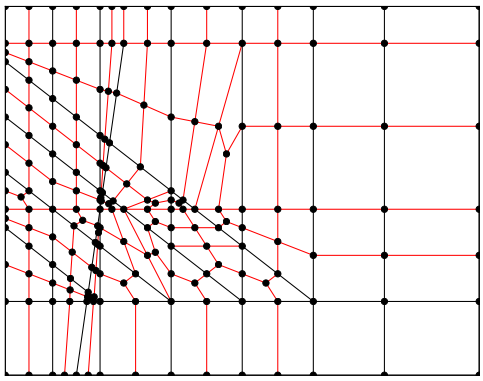
(b) Intersections computed



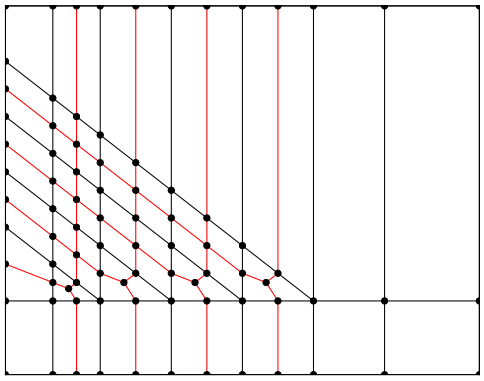
(c) Connecting edges added



(d) Pentagons split



(e) Final result



(f) Final result with one edge removed

Figure 8: Snapshots at different stages of the skin layout generation process.

number of quads, which is undesirable; however, this is due to the initial distribution of edges. If the one irregular edge is removed from 8(a), the final result has much fewer quadrilaterals as shown in 8(f).

B. Internal Structure Generation

With the skin layout computed, each quadrilateral can be divided into shell elements with the desired mesh resolution. The nodes are projected onto the OML and the B-spline basis functions are evaluated at the projected points to obtain a mapping from the OML control points to the structural surface mesh. The next step is to find the mapping from the structural surface mesh to the full structural mesh. This means that each node in the internal structure must be parametrized in terms of one or more nodes on the skin.

The approach is to specify the desired internal structure in a 3-D volume, whose top and bottom faces are mapped to parts of the structural surface mesh. The details are best explained through an example, which is given in Figure 9. The wing structure on the far right has ribs and two main spars with holes removed as well as stiffeners for the upper and lower skins with L-shaped cross-sections. The parametric generation of complex structures such as this is greatly simplified using a 3-step process. First, a list of (u_0, v_0) coordinates are computed for all nodes in each member: for example, a square with a round hole. Next, these (u_0, v_0) coordinates are mapped to (u, v, w)-space based on the specified structural layout for the component. This involves the evaluation of a bilinear interpolant at locations given by (u_0, v_0) and corner vertices given by the coordinates of the element's corners in (u, v, w)-space. Finally, the (u, v, w) coordinates are mapped to the (x, y, z) coordinates in physical space. To do this, the four nearest nodes in the skin layout are found for each internal node. The parametric coordinates of (u, v) within the quadrilateral formed by these four nearest nodes are then computed; this amounts to an inverse bilinear mapping, which has an analytic solution involving the quadratic equation. The computed parametric coordinates are used to perform a forward evaluation of the same bilinear interpolant, but using the coordinates of the node on the upper and lower skins in physical space as opposed to (u,v,w)-space. The final (x, y, z) coordinate is the weighted average of the evaluated bilinear interpolants for the upper and lower skins where w is the weight. The final coefficients are assembled in a Jacobian matrix and stored, after which obtaining the full set of structural nodes from an updated set of OML B-spline control points reduces to just a simple matrix-vector product.

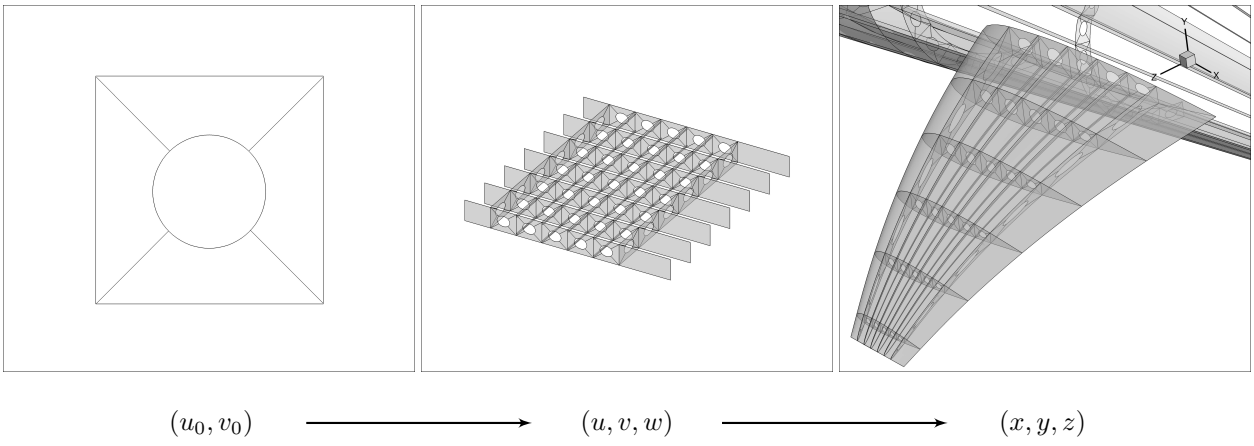


Figure 9: The three stages of the internal structure generation process.

The mapping of a structural layout in (u, v, w)-space to (x, y, z)-space is intuitive, but there are two special cases. The first occurs when either the upper or lower skin where the interior node is located does not exist because the node is located near a wing-nacelle junction or some other interface between components. In this case, the missing portion of the skin must be approximated using the basic transfinite interpolation formula, which generates a surface from its bounding edges. The second is fuselage structures. In this case, the upper skin is the fuselage surface and the lower skin is actually the average of the top- and bottom-most curves of the fuselage surface: i.e. the lower skin is just a curve instead of a surface. This has the effect that the mapping transforms to a cylindrical coordinate system where w maps to the radial coordinate, v maps to the tangential coordinate, and u is the axial coordinate. Using this approach, fuselage frames and longerons can be modeled with the same amount of detail including patches with holes as shown in Figure 10.

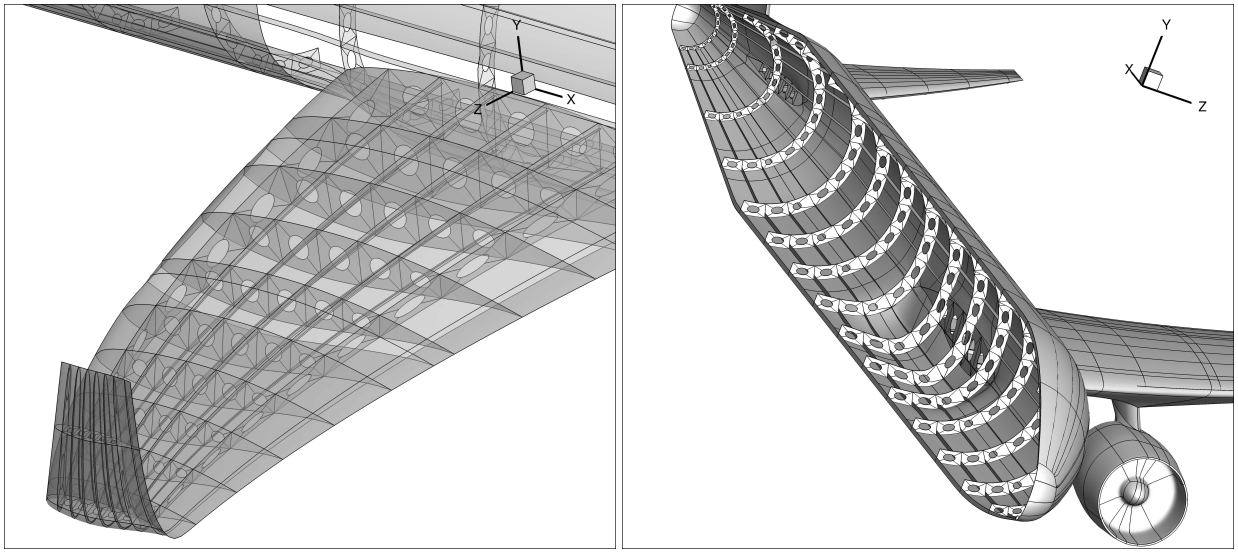


Figure 10: Wing and fuselage structures for the conventional configuration.

Figure 11 shows the conventional configuration with structures for the fuselage, wing, and tail modeled. The code for specifying the fuselage and wing structures in the conventional aircraft is shown below.

```
c['fuse'].addMembers('Longerons', 2, 1, 12, 15, A1=[0,0,0.95], C1=[1,0,1], A2=[0,1,0.95], C2
    =[1,1,1])
c['fuse'].addMembers('Frames', 2, 2, 16, 11, A1=[0,0,0.85], C1=[0,1,1], A2=[1,0,0.85], C2
    =[1,1,1])
```

```
c['wing'].addMembers('RibsLE', 1, 1, 13, 1, A1=[0,0,0], C1=[0,0.125,1], A2=[1,0,0], C2
    =[1,0.125,1])
c['wing'].addMembers('Ribs', 1, 2, 13, 5, A1=[0,0.125,0], C1=[0,0.75,1], A2=[1,0.125,0], C2
    =[1,0.75,1])
c['wing'].addMembers('RibsTE', 1, 1, 13, 1, A1=[0,0.75,0], C1=[0,1,1], A2=[1,0.75,0], C2=[1,1,1])
c['wing'].addMembers('Spars', 1, 2, 2, 12, A1=[0,0.125,0], C1=[1,0.125,1], A2=[0,0.75,0], C2
    =[1,0.75,1])
c['wing'].addMembers('Ustiff', 1, 1, 4, 12, A1=[0,0.25,0.9], C1=[1,0.25,1], A2=[0,0.625,0.9], C2
    =[1,0.625,1])
c['wing'].addMembers('UstiffL', 1, 1, 4, 12, A1=[0,0.25,0.9], B1=[0,0.255,0.9], C1=[1,0.255,0.9],
     D1=[1,0.25,0.9], A2=[0,0.625,0.9], B2=[0,0.63,0.9], C2=[1,0.63,0.9], D2=[1,0.625,0.9])
c['wing'].addMembers('Lstiff', 1, 1, 4, 12, A1=[0,0.25,0], C1=[1,0.25,0.1], A2=[0,0.625,0], C2
    =[1,0.625,0.1])
c['wing'].addMembers('LstiffL', 1, 1, 4, 12, A1=[0,0.25,0.1], B1=[0,0.255,0.1], C1=[1,0.255,0.1],
     D1=[1,0.25,0.1], A2=[0,0.625,0.1], B2=[0,0.63,0.1], C2=[1,0.63,0.1], D2=[1,0.625,0.1])
```

A significant challenge for modeling the entire structure of an airplane with multiple components is that the number of edges may not match at locations where components intersect. The proposed solution is to use the junction components as a buffer. O-gridding and other similar concepts can be applied to overcome the fact that the number of edges entering the junction from one side differs from the number of edges entering from the opposite side. There are other limitations of the current version of the structural modeler, including the inability to model the extension of the wingbox to the symmetry plane. Work is currently being done to address this and other limitations.

VI. Conclusion

The premise of this paper is that the commercial aircraft design process stands to benefit tremendously if high-fidelity single- and multi-disciplinary optimization with many variables can be properly leveraged. State-of-the-art optimization algorithms are capable of handling hundreds, up to thousands of design variables with multiple disciplines

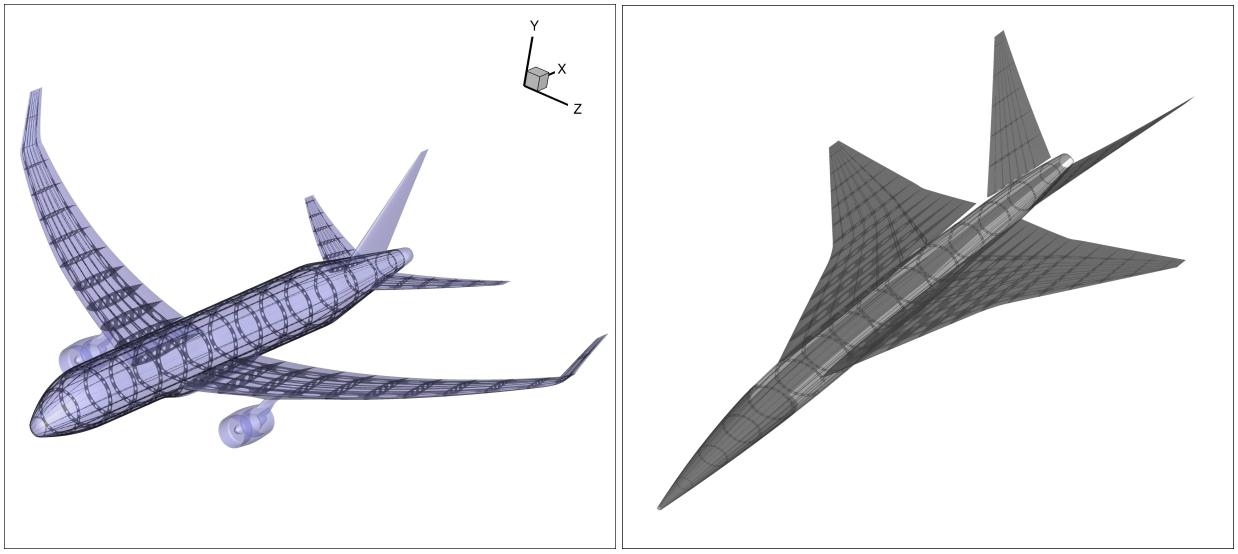


Figure 11: Structures shown for the full conventional and supersonic configurations.

coupled. Yet, their applicability to practical design is hindered by the lack of a configuration-level geometry modeler and a framework to handle integration of codes. Many aircraft design tools and geometry engines can be found in the literature, but existing frameworks are not capable of supporting high-fidelity MDO with hundreds of design variables. The GeoMACH tool suite addresses this need through three contributions.

The first is the design of the framework itself and the approach to MDAO. By taking a geometry-centric approach to MDAO, GeoMACH allows both the CFD and structural meshes to be driven by a parametrically defined OML. Furthermore, this mapping from the OML to the meshes can be reused for data transfer between disciplines as shown in Figure 2. This is possible because of GeoMACH's efficient, lightweight B-spline engine. Leveraging the OpenMDAO framework, the end result is a modular, solver-independent tool suite that also facilitates derivative computation.

The second is GeoMACH's parametric OML modeler, which has the unique ability to span a configuration-level design space while modeling the OML with untrimmed 4-sided surfaces. This component is what enables GeoMACH to efficiently perform optimization with a large number of design variables through support for structured CFD and mesh movement. The OML modeler supports both high-level control with a small number of aircraft parameters as well as more precise control, while allowing for large shape changes required in conceptual design.

Finally, GeoMACH's parametric structural modeler is another unique tool that also provides detailed modeling capability using a simple and intuitive interface. It can model details such as longerons and frames with holes in the fuselage as well as spars, ribs, and stringers with L-shaped sections in wing-type components. The structural modeler assembles a Jacobian matrix mapping the structural surface mesh to the full structural mesh, enabling the structural model to be parametrically controlled by the OML through an inexpensive linear transformation.

VII. Acknowledgments

This work is supported by NASA through award No. NNX11AI19A. The authors would like to thank Justin Gray, Kenneth Moore, and Christopher Heath from NASA Glenn Research Center as well as Robert Haimes from MIT for their helpful discussions. The authors would also like to thank GaRam Jun and Gaetan Kenway for their help with creating some of the aircraft models and performing CFD analysis.

References

[1] Hicken, J. E. and Zingg, D. W., "Induced-Drag Minimization of Nonplanar Geometries Based on the Euler Equations," *AIAA Journal*, Vol. 48, No. 11, 2010, pp. 2564–2575. doi:10.2514/1.52436.

[2] Mader, C. A. and Martins, J. R. R. A., "Optimal Flying Wings: A Numerical Optimization Study," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, HI, April 2012.

[3] Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A., "A Scalable Parallel Approach for High-Fidelity Aerostructural Analysis and Optimization," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, HI, April 2012, AIAA 2012-1922.

[4] Kennedy, G. J. and Martins, J. R. R. A., "Aerostructural design optimization of composite aircraft with stress and local buckling constraints using an implicit structural parametrization," *Proceedings of CASI AERO 2011*, Montreal, Quebec, April 2011.

[5] Heath, C. and Gray, J., "OpenMDAO: Framework for Flexible Multidisciplinary Design, Analysis and Optimization Methods," *Proceedings of the 53rd AIAA Structures, Structural Dynamics and Materials Conference*, Honolulu, HI, April 2012, AIAA-2012-1673.

[6] Martins, J. R. R. A. and Hwang, J. T., "Review and Unification of Methods for Computing Derivatives of Multidisciplinary Systems," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, HI, April 2012, AIAA 2012-1589.

[7] Hahn, A. S., "Vehicle Sketch Pad: A Parametric Geometry Modeler for Conceptual Aircraft Design," *Proceedings of the 48th AIAA Aerospace Sciences Meeting*, Orlando, FL, January 2010, AIAA-2010-0657.

[8] Risse, K., Anton, E., Lammering, T., Franz, K., and Hoernschemeyer, R., "An Integrated Environment for Preliminary Aircraft Design and Optimization," *Proceedings of the 53rd AIAA Structures, Structural Dynamics and Materials Conference*, Honolulu, HI, April 2012, AIAA-2012-1675.

[9] Rodriguez, D. L. and Sturdza, P., "A Rapid Geometry Engine for Preliminary Aircraft Design," *Proceedings of the 44th AIAA Aerospace Sciences Meeting*, Reno, NV, January 2006, AIAA-2006-0929.

[10] LLC, A., "AVID PAGE," `http://www.avidaerospace.com/software/avid-page`, Accessed July 11, 2012.

[11] Hahn, A. S., "Application of Cart3D to Complex Propulsion-Airframe Integration with Vehicle Sketch Pad," *Proceedings of the 50th AIAA Aerospace Sciences Meeting*, Nashville, TN, January 2012, AIAA-2012-0547.

[12] Chaput, A. J. and Rizo-Patron, S., "Vehicle Sketch Pad Structural Analysis Module Enhancements for Wing Design," *Proceedings of the 50th AIAA Aerospace Sciences Meeting*, Nashville, TN, January 2012, AIAA-2012-0546.

[13] Gray, J., Moore, K. T., Hearn, T. A., and Naylor, B. A., "A Standard Platform for Testing and Comparison of MDAO Architectures," *Proceedings of the 53rd AIAA Structures, Structural Dynamics and Materials Conference*, Honolulu, HI, April 2012, AIAA-2012-1586.

[14] Zingg, D. W., Nemec, M., and Pulliam, T. H., "A Comparative Evaluation of Genetic and Gradient-Based Algorithms Applied to Aerodynamic Optimization," *Shape design in aerodynamics REMN – 17/2008.*, 2008, pp. 103–126.

[15] 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*, Fort Worth, TX, September 2010, AIAA 2010-9231.

[16] Hicken, J. and Zingg, D., "Aerodynamic Optimization Algorithm with Integrated Geometry Parameterization and Mesh Movement," *AIAA Journal*, Vol. 48, No. 2, February 2009, pp. 400–413.

[17] Hwang, J. T., Waas, A. M., and Martins, J. R. R. A., "Micromechanical Modeling and Design Optimization of 2-D Triaxial Braided Composites," *Proceedings of the 50th AIAA Aerospace Sciences Meeting*, Nashville, TN, January 2012, AIAA 2012-1257.

[18] Brown, S. A., "Displacement Extrapolation for CFD+CSM Aeroelastic Analysis," *Proceedings of the 35th AIAA Aerospace Sciences Meeting, Reno, NV*, 1997, AIAA 1997-1090.

[19] de Boor, C., *A Practical Guide to Splines*, Springer, 2001.