21st AIAA Computational Fluid Dynamics Conference
24 - 26 June 2013, San Diego, California

AIAA 2013-2850

Geometry Generation of Complex Unconventional Aircraft with Application to High-Fidelity Aerodynamic Shape Optimization

Hugo Gagnon* and David W. Zingg†

Institute for Aerospace Studies, University of Toronto, Toronto, Ontario, M3H 5T6, Canada

This paper describes a geometry generator capable of providing a high-quality model of unconventional aerospace vehicles with fast turnaround. These desirable but contradictory properties are made possible by striking the right balance between user interaction and modeling automation. The former is achieved through a popular Python-based command line interface and the latter through a component-based approach. Final geometries are defined analytically with watertight networks of non-uniform rational B-spline surfaces that are topologically four-sided. Such a tool is expected to leverage the high-fidelity multidisciplinary design analysis and optimization packages that are being increasingly incorporated in earlier stages in the design of novel aircraft. As an example, a drag minimization problem applied to a regional jet geometry generated by the proposed tool concludes the paper.

I. Introduction

THORETICALLY speaking, unconventional configurations have more potential than their ubiquitous tube-and-wing counterparts in terms of reducing drag and thus fuel consumption. In his preliminary studies on nonplanar concepts, Kroo states that wings of height-to-span ratio of 0.2 can reduce induced drag by as much as 30%.[1] On a more practical note, Liebeck reports the blended-wing-body air carrier concept to offer fuel burn savings per seat mile in the order of 27% compared with the conventional baseline.[2] Clearly, many other design alternatives with potentially even higher benefits exist. High-fidelity Aerodynamic Shape Optimization (ASO) as well as Multidisciplinary Design Optimization (MDO) will play a crucial role in the design of such configurations, and at progressively earlier stages of the design process.

An obvious choice for generating high-fidelity geometries is offered by the many commercially available Computer-Aided Design (CAD) packages. While their modeling capabilities are seemingly limitless, their usefulness to conceptual and even preliminary design is far from ideal. To begin with, CAD packages inevitably require skilled operators. This is because the outer mold line of aircraft does not typically result from primitive CAD operations but rather through a series of meticulously designed free-form surfaces. It is however unreasonable to expect the conceptual designer, who is usually not a CAD specialist, to surmount the steep learning curve associated with those packages all the while meeting strict time constraints. Even in the best case scenario, the minutiae involved in conventional CAD approaches can be an actual hindrance to the fast-paced, highly cyclical nature of conceptual design. Last but not least, proprietary rights protecting CAD kernels make it close to impossible to compute analytical sensitivities through say the adjoint formulation, an extremely limiting factor for gradient-based ASO and MDO. While external interfaces can partly alleviate those issues,[3–5] customizing them can be just as difficult as developing a brand new, tailored geometry modeler.

For these and other reasons, a number of conceptual aircraft design tools have seen light over the past few years.[6–13] VSP,[6] developed by the Aeronautics Systems Analysis Branch at NASA Langley Research Center, is based on text files and excels at parametric geometry definition, but has limited support for arbitrary shapes. Desktop Aeronautics' RAGE[8] also uses scripts while focusing on a component-based approach;

*PhD Candidate and AIAA Student Member

†Professor and Director, Tier 1 Canada Research Chair in Computational Aerodynamics and Environmentally Friendly Aircraft Design, J. Armand Bombardier Foundation Chair in Aerospace Flight, AIAA Associate Fellow

Copyright © 2013 by Hugo Gagnon and David Zingg. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission.

generic components are designed separately and subsequently assembled. This works well, but then surface triangulation or some other ad hoc treatment becomes necessary in order to determine intersections which, in the context of gradient-based shape optimization, may in turn introduce noise in surface sensitivities or even preclude the use of a mesh movement algorithm altogether. What is more, if required an optimized geometry still has to be translated into a CAD format. A tool that overcomes those limitations is GeoMACH,[9] currently developed at the University of Michigan's MDO Laboratory and part of the broader scoped OpenMDAO initiative.[14] It is worth noting that, although all the aforementioned applications offer some kind of visual inspection capabilities, only a selected few give the designer full freedom to pick and drag geometric entities.

In contrast, we present here an interactive-centred geometry generator, hereafter referred to as GENAIR, that is a hybrid between the script-based tools just described and a CAD package. It is written purely in Python for maximum compatibility, from core to interface, the latter of which simply reuses the powerful command line utility *IPython*.[15] From it, a designer can instantiate any number of `Airfoil`, `Wing`, `Fuselage`, etc. objects and visualize them on the fly by passing them as arguments to a generic `draw` function. He or she can then of course inspect the displayed objects but also modify them interactively through mouse and keyboard inputs. The result is a truly interactive design environment, albeit minimalistic, that is intended to be natural to the aircraft designer and where most mundane tasks are automated. Just like any modern CAD package, GENAIR's geometry engine relies entirely on non-uniform rational B-splines (NURBS) objects.[16] This choice not only guarantees geometries to be represented to any desired level of accuracy, but it also reflects recommendations from the NASA-IGES NURBS-Only data exchange specification standard.[17]

The other "raison d'être" of GENAIR is to integrate with shape optimization packages. As such it could serve as a basis for a geometry-centric MDO framework[9] by 1), facilitating communication among disciplines and 2), automatically regenerating components following changes in the geometric design variables. The design variables could be anything from individual coordinates of NURBS control points to component-based parameters, such as a wing's dihedral or its position relative to a fuselage. In the present work, however, we use GENAIR exclusively as a standalone tool for the initial setup of a geometry; during the actual optimization — which, unlike GENAIR, is fully automated — both the geometry and the computational mesh are controlled by a robust two-level free-form deformation approach,[18] while an efficient adjoint-based algorithm evaluates the functional and its gradient with respect to the design variables.[19]

The remainder of this paper is divided as follows. Section II describes GENAIR's unique design environment, including its lightweight OpenGL-based windowing system. The basic mathematical concepts behind NURBS as well as an overview of GENAIR's own NURBS library are covered in Section III. Then, the basic aircraft components, each encapsulated in their own Python class, are individually dissected in Section IV. After introducing our in-house aerodynamic optimizer in the first half of Section V, in the second half we present a lift-constrained drag minimization problem applied to a conventional regional jet built with GENAIR. Finally, concluding remarks and future directions are given in Section VI.

II. IPython/OpenGL User Interface

Design is by nature an interactive process. Python, an easy-to-use, object-oriented scripting language that is now gaining widespread popularity among the scientific community, has plenty of third-party packages to accomodate interactivity, one of which is *IPython*[15] (the "I" in fact stands for "Interactive"). It is essentially an augmented Python shell that, when used in conjunction with the powerful *NumPy* (Numerical Python) and *SciPy* (Scientific Python) libraries,[20] can offer similar, if not superior, functionality to Matlab's computing environment.

IPython's philosophy is well in line with GENAIR's, which is to provide the user an uncluttered yet highly productive interface. Its features include tab completion and object introspection, as well as debugging and profiling tools, to name just a few. As an example of object introspection, issuing the command `Airfoil?` at the prompt returns a detailed description of what this object does along with its intended usage. *IPython* is also highly configurable, and as such ships with predefined shell aliases and an extensible "magic" function system. A magic function is one that is preceded by the percent character; for example, typing `%cd` changes the current directory to the user's home directory.

Upon launching, GENAIR automatically loads its most commonly used aircraft-related Python classes, e.g. `Airfoil` and `Wing`, and injects them directly into the user's *IPython* namespace for ease of access. Any number of those (or any other Python objects for that matter) can then be instantiated on the fly as required by the design process; however such objects will only remain alive until the current *IPython* session

is terminated. In order to prevent the user from losing work, GENAIR provides the opportunity to save to disk, at any moment, any number of any objects by means of a simple `save` function. Naturally, there is a complementary `load` function that restores previously saved objects. Both functions are thin wrappers built around the very efficient *cPickle* module, part of the standard Python library.

The command line interface suffices to generate a wide range of geometries; in fact, with the exception of the `Junction` (to be discussed in Section IV), all aircraft components can be created solely from it. It could thus just as well be used in batch mode. However, GENAIR's true strength resides in its ability to not only display objects for inspection purposes, but also to allow the designer to modify them in real time, by either moving around individual control points of NURBS objects or, if need be, their entire control lattices. This feature is extremely useful when designing smooth `Fuselage` profiles or the trajectory curve along which an `Airfoil` is swept to create an exotic `Wing`.

Any number of three-dimensional points and NURBS objects can be drawn in the same window, and there can be multiple windows running concurrently. Internally, each window is handled by *pyglet*.[21] On top of constantly querying the operating system for new user input, *pyglet* supplies each window its own OpenGL context, which can be spawned or destroyed on the fly. While *pyglet* is in itself a port to OpenGL bindings, in some cases GENAIR relies on *PyOpenGL*[22] to circumvent the need for *ctypes*. For example, with *PyOpenGL* a NURBS curve can be rendered by simply passing the references of the *NumPy* arrays that define it directly to the GLU function wrapper, in this case `gluNURBSCurve`. Finally, note that in order to prevent the windows from blocking the *IPython* prompt, the main *pyglet* event loop is explicitly rewritten and directly hooked with the underlying Python C facility `PyOS_InputHook`.

III. NURBS Library

The popularity of NURBS in the CAD and graphics communities partly follows from their incorporation in national and international standards such as IGES,[17,23] but is mostly due to their excellent properties. For example, NURBS can represent both analytic (conics, circles, surfaces of revolution, etc.) and free-form entities precisely and compactly. They also offer a clear geometric interpretation; by manipulating their control points and weights a designer can intuitively modify their shapes. In the context of ASO, their high flexibility and efficient approximation power make them prime candidates when no a priori knowledge of the final geometry is known. Other advantages of NURBS include affine invariance, local support, and the strong convex hull property.[16]

Formally, a NURBS curve of degree p is defined as[16]

$$\mathbf{C}(u) = \sum_{i=0}^{n} \mathcal{R}_i^{(p)}(u)\,\mathbf{P}_i, \qquad a \le u \le b. \tag{1}$$

A designer is free to choose the number, $n+1$, and location of the control points $\{\mathbf{P}_i \in \mathbb{R}^3,\ i = 0, \ldots, n\}$, as well as the degree, p, and knot vector,

$$\mathbf{U} = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_n, \underbrace{b, \ldots, b}_{p+1}\},$$

with which the rational basis functions,

$$\mathcal{R}_i^{(p)}(u) = \frac{\mathcal{N}_i^{(p)}(u)\,w_i}{\sum_{j=0}^{n} \mathcal{N}_j^{(p)}(u)\,w_j},$$

are defined. These are themselves composed of user-defined scalar weights $\{w_i\}_{i=0}^{n}$ and pth-degree B-spline functions $\{\mathcal{N}_i^{(p)}(u)\}_{i=0}^{n}$. The piecewise polynomials making up these splines are joined at the non-uniform and non-decreasing knot locations $\{u_i\}_{i=0}^{n+p+1}$ introduced above, such that the basis is C^{p-k} continuous at a given knot, where k is its multiplicity. Note that we choose a non-periodic knot vector, i.e. with end knot multiplicities equal to the order of the splines, which ensures that $\mathbf{C}$ passes exactly through its end points $\mathbf{P}_0$ and $\mathbf{P}_n$. This property becomes immediately apparent after examining the recursive relationships defining

B-splines:

$$\mathcal{N}_i^{(0)}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1}, \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathcal{N}_i^{(p)}(u) = \frac{u - u_i}{u_{i+p} - u_i}\mathcal{N}_i^{(p-1)}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}}\mathcal{N}_{i+1}^{(p-1)}(u)\,.$$

Similarly, a NURBS surface (or patch) can be created using tensor-products:

$$\mathbf{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m}\mathcal{N}_i^{(p)}(u)\,\mathcal{N}_j^{(q)}(v)\,\mathbf{P}_{i,j}^w, \tag{2}$$

where the indices p and n respectively refer to the degree and last control point in the u direction; similar definitions hold for the v direction. Note that here $\mathbf{S}^w(u,v)$ is defined in four-dimensional homogeneous space (X,Y,Z,W) in order to offer a non-rational, compact analogue to Eq. (1) that is considerably easier to implement. Assuming all weights to be nonzero, projecting the homogeneous control points $\{\mathbf{P}_{i,j}^w = (wx, wy, wz, w)_{i,j} \in \mathbb{R}^4,\ i = 0,\ldots,n,\ j = 0,\ldots,m\}$ back onto the $W = 1$ plane retrieves the desired three-dimensional surface $\mathbf{S}(u,v)$. For the special case where all weights are set to unity, it can be shown that $\mathbf{S}^w(u,v)$ simply reduces to $\mathbf{S}(u,v)$, i.e. to the traditional B-spline surface definition. Thus, without loss of generality, all weights are now assumed to be unity, and the superscript w is dropped in the remainder of the manuscript.

Due to the unavailability of a reliable, non-proprietary NURBS library (a notable exception being OpenCASCADE[24]), a simple yet complete one has been developed from scratch. At its lowest level it is composed of the five fundamental geometric algorithms: knot insertion, knot refinement, knot removal, degree elevation and degree reduction.[25] On top of these are built higher-level functions and utilities which GENAIR's top-level classes heavily rely on; they will be described next in the context of component-based aircraft design. Note that at times the designer may also find it useful to interact with the NURBS library directly. This task is simplified by the fact that it is coded, just like everything else in GENAIR, in pure Python. This would normally be impractical due to the prohibitive cost of some CAD operations, but we found that by heavily exploiting *NumPy*'s vectorization capabilities such as broadcasting,[20] it is possible to reach near compiled-language speeds. This should come as no big surprise since most of *NumPy* is in fact interfaced with optimized, pre-compiled C functions.

IV. Component-Based Aircraft Design

Just like most tools of its kind,[6–10,12,13] GENAIR adopts a component-based approach to aircraft design. This design philosophy is built on the premise that most aircraft can be built from basic components that may be subsequently assembled in some arbitrary manner. GENAIR predefines several such components, the most important ones being the `Fuselage`, `Wing` and `Junction` classes. As an example applied to a conventional aircraft, a designer would typically start by forming three `Wing` objects, one for the main wing and two for the empennage, and position them relative to a `Fuselage`. He or she would then seal all surfaces with as many `Junction`s as there are intersecting components, in this case three. At this point the designer could perhaps add a `Nacelle` and another `Wing` as a pylon, or just go ahead and output the geometry in IGES format for meshing purposes.

GENAIR also permits a previously designed component to be automatically regenerated after modifying one of its key defining parameters. Identifying such parameters is natural since GENAIR captures the user's design intent, in contrast to a CAD operator's interpretation of it. This opens up the door to component-based ASO and MDO, in which a design variable could be, for example, the position of a `Wing` relative to a `Fuselage`. Now, there may very well be hundreds of other design variables involved, on top of the millions of state variables that are inherent to high-fidelity disciplinary solvers. Such large-scale problems are currently only amenable through adjoint-based algorithms, which in turn usually rely on smooth mesh deformations so as to ensure continuous design spaces (see Ref. 26 where this requirement is alleviated for Cartesian grids). GENAIR provides the necessary foundation to do just that, partly by guaranteeing that the network of surfaces defining a geometry will always be watertight and, if need be, topologically four-sided. As one might expect, of all the components it is the `Junction` that is most affected by those requirements.

Next, we describe in some detail each one of GENAIR's three main components, plus the `Nacelle` and `Wingstructure`. Refer to Figures 1 and 2 where they are respectively applied to the design of a conventional utility jet and unconventional aircraft with disparate features. More descriptive drawings of the unconventional designs can be found in Appendix A. In general, because aerodynamics is sensitive to the curvature of the surface, special emphasis is placed on C^2 or even higher continuity, except across edges shared by two adjacent surfaces where at the very least G^1 continuity is enforced. This is with the exception of wing-body junctions and the like, which we deliberately assume to be C^0 continuous. Also, as already discussed in the previous paragraph, since the same surfaces are to be reused in a three-dimensional, multi-block structured grid setting, further care is devoted to domain decomposition, maintaining "good" parameterizations and avoiding nondegeneracies. Finally, note that the following discussion focuses, mostly from a CAD perspective, on the underlying implementation of each component. As far as the end user is concerned these details are often irrelevant.

A. `Fuselage`

A widely popular approach to designing the outer mold line of a fuselage is to start by creating a series of cross-section curves that are subsequently skinned (lofted) with an interpolant. Depending on the desired final shape, this process is often tedious — start with a small number of curves, skin, take plane cuts, modify new curves, repeat — and can thus become extremely time consuming. Therefore, we instead opt for a top-down approach where the `Fuselage` initially assumes the shape of a bullet-head half-cylinder. Analogous to clay, the idea is to progressively deform this shell by applying a series of so-called "molders", better known as Free-Form Deformation (FFD) volumes.[27] For example, two `FuselageMolders` are normally used to independently design the nose and rear parts of the `Fuselage`, while only one `FairingMolder` is usually adequate to emulate say a wing-body fairing. See Figure 1(a) where a `FuselageMolder` (light blue) is applied on the aircraft's nose.

We use trivariate NURBS to act as FFD molders. To ease their manipulations we associate their control lattice with a set of "pilot points" (the empty black squares in Figure 1(a)). Depending on the type of molder employed, their behavior is different but the goal is always the same: to transform groups of control points pertaining to the same lattice together as opposed to individually. This technique allows for the creation of well-behaved aerodynamic shapes in a matter of minutes, and lends itself quite well for shape optimization purposes where pilot points may become design variables.

B. `Wing`

A `Wing` is instantiated from a root `Airfoil` and, optionally, a tip `Airfoil`. If no tip `Airfoil` is specified, the `Wing` will assume the shape of the root `Airfoil` only; otherwise its shape will result from a linear interpolation between the root and tip `Airfoil`s. Each `Airfoil` is characterized by a B-spline curve $\mathbf{A}(u)$, which is obtained from fitting a set of ordered data points $\{\mathbf{D}_k \in \mathbb{R}^2,\ k = 0, \ldots, m\}$ with as few control points as possible and to within a specified error bound:[25]

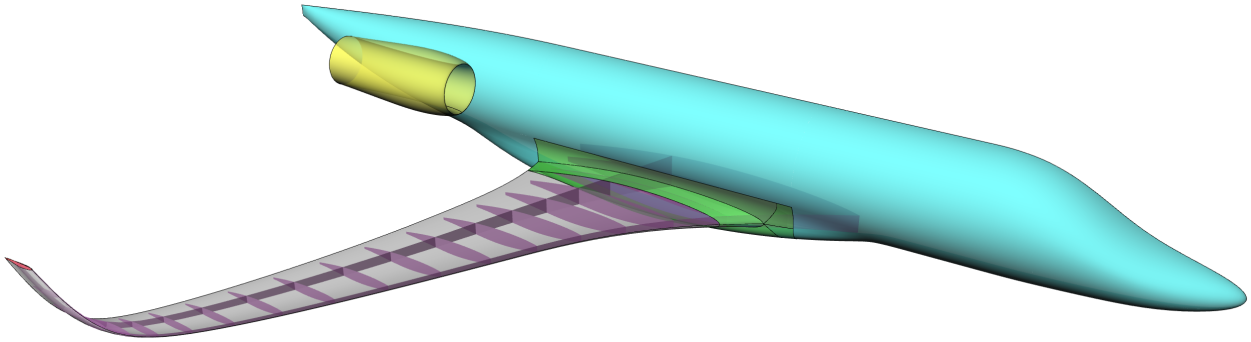
$$\max_{0 \le k \le m} \left(\min_{a \le u \le b} \|\mathbf{D}_k - \mathbf{A}(u)\| \right).$$

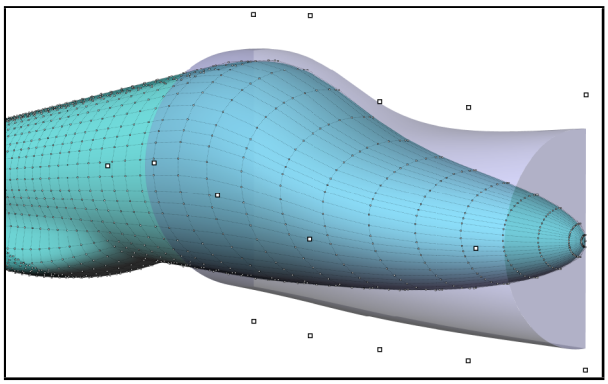
Should an `Airfoil` be blunt it is possible to subsequently sharpen it with a curvature-continuous extension,[28] such as would be required for an inviscid-flow computation.

The `Wing`'s skin is generated by sweeping $\mathbf{A}(u) = \mathbf{A}_{\text{root}}(u)$ (or $(1-v)\mathbf{A}_{\text{root}}(u) + v\mathbf{A}_{\text{tip}}(u)$ if the `Wing`'s root and tip `Airfoil`s differ) along a user-defined trajectory curve $\mathbf{T}(v)$:[25]

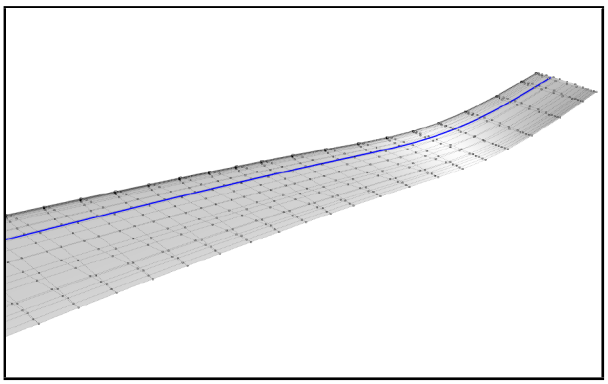
$$\mathbf{W}(u, v) = \mathbf{T}(v) + \mathsf{G}(v)\mathsf{S}(v)\mathbf{A}(u),$$

where $\mathsf{S}(v)$ and $\mathsf{G}(v)$ are scaling and general transformation matrices, respectively. Unlike $\mathsf{S}(v)$, which defaults to the identity matrix, $\mathsf{G}(v)$ is in general not representable in closed NURBS form. Therefore, an approximate surface $\widehat{\mathbf{W}}(u, v)$ is constructed by first transforming $K + 1$ instances of $\mathbf{A}(u)$ along $\mathbf{T}(v)$ and skinning across them. Thus, the trajectory's degree q and knot vector $\mathbf{V}$ are inherited by $\widehat{\mathbf{W}}(u, v)$, and its accuracy can be increased by increasing K. Refer to Figure 1(b) where in this case the trajectory curve is located at the quarter chord.

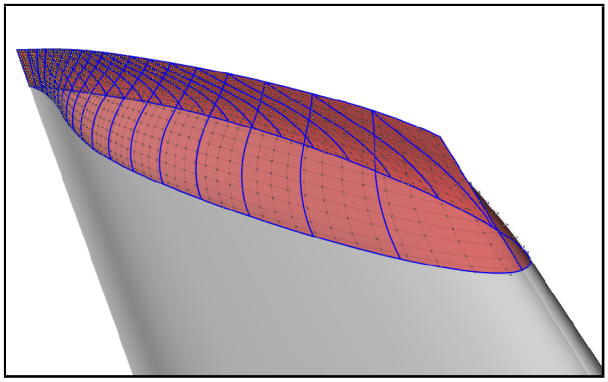




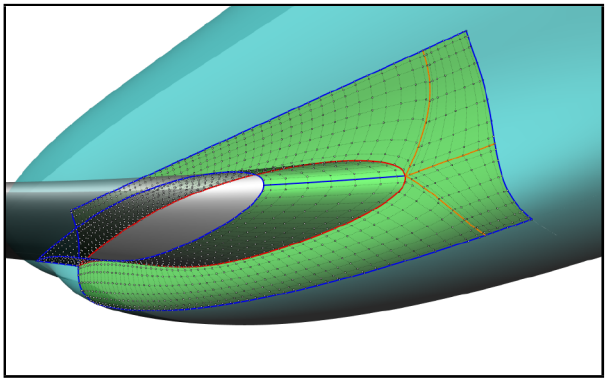
(a) `Fuselage`



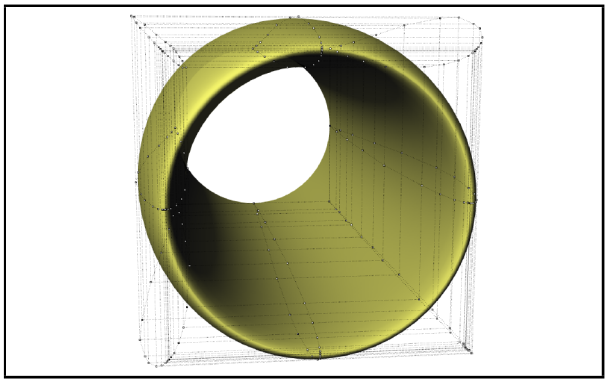
(b) `Wing`



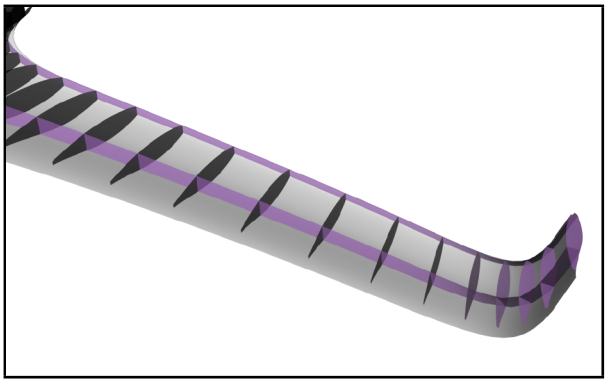
(c) `Wingtip`



(d) `Junction`



(e) `Nacelle`



(f) `Wingstructure`

Figure 1. Component-based aircraft design.

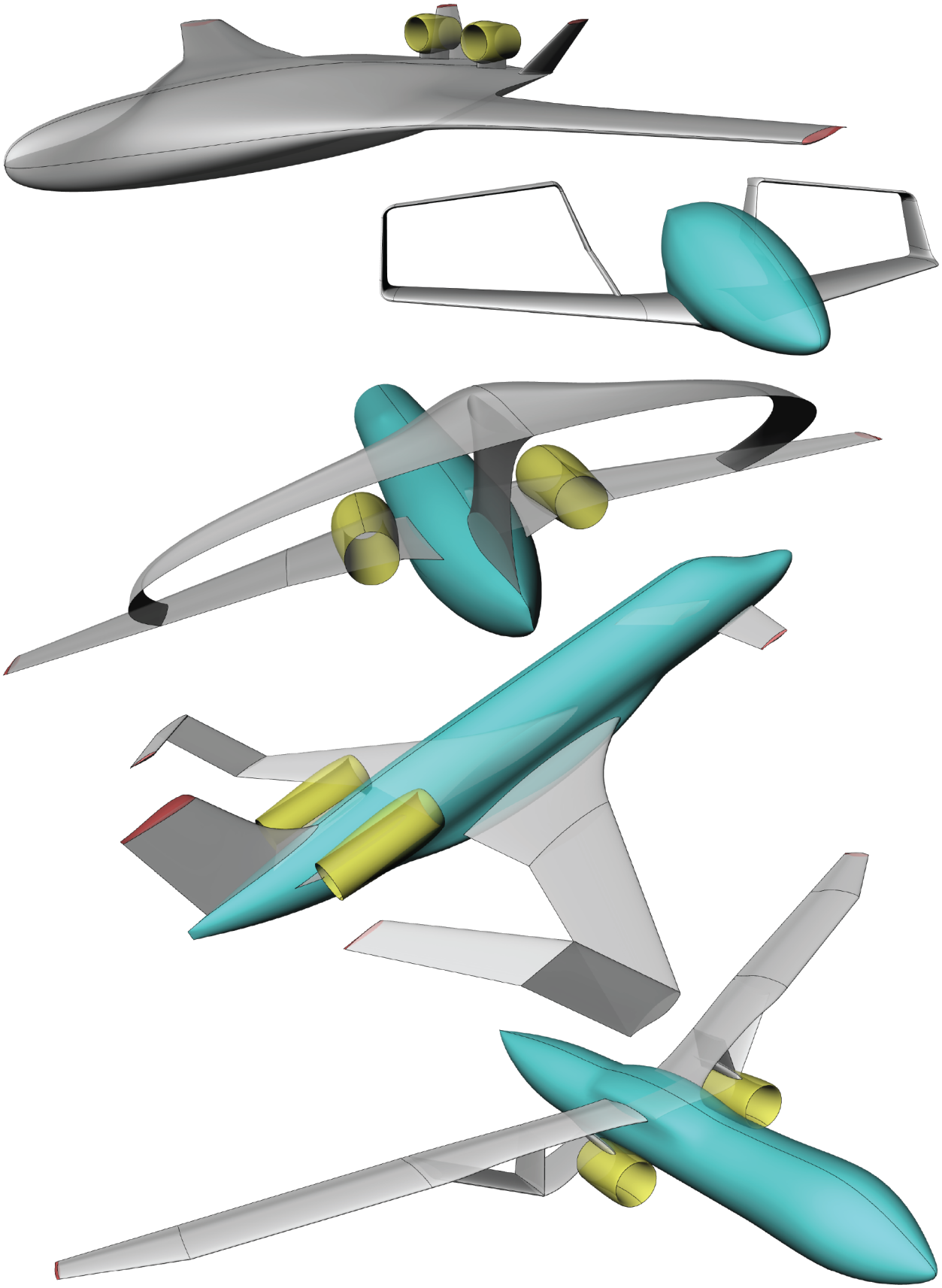


Figure 2. Unconventional aircraft built from generic components.

A single `Wing` is usually sufficient when generating say blended-wing-bodies, elliptical wings, ring-wing systems, or in fact any other arbitrary wing whose spanwise variation is smooth. However, for more complicated shapes involving say cranked or segmented C-wings such as those found in Figure 2, it is possible to attach two or more `Wing` objects together. The `Wing`s can be of arbitrary size and shape, the only requirement being that the two `Airfoil`s meeting at one end must match. Note that this may incur a slight reapproximation of the surfaces in the immediate vicinity of the merged area as an exact intersection curve there does not, in general, exist.

Finally, to close the gap at the end of a `Wing`, GENAIR provides a library of `Wingtip`s to choose from, including rounded and raked wingtips. Their construction generally involves interpolating an automatically generated bidirectional curve network (visible in blue in Figure 1(c)), a technique due to Gordon.[29] This process is completely transparent to the designer, who is free to vary any of the available shape parameters as he or she sees fit. Currently, among those options are the number of curves employed while forming the Gordon network and the spanwise length of the `Wingtip` extension itself.

C. `Junction`

Once the relative position between a `Wing` and a `Fuselage` is fixed, or in fact between a `Wing` and any other component, the question of interest is: how does one obtain watertight surfaces from two intersecting surfaces? Trimming them is not an option since, in general, an intersection curve cannot be determined analytically[a]. Therefore, given that relatively new CAD technologies such as T-splines[30] have yet to mature, some kind of reapproximation scheme must be performed. The first step in doing so is typically to isolate the intersection by extracting one patch from each surface. At this point one could simply discard both patches and close the hole so produced by interpolating the boundaries of the remaining surfaces; this is the approach taken by GeoMACH,[9] and is arguably the most efficient one. In contrast, we choose to trade efficiency for quality by closely reapproximating the extracted patches with a network of four-sided surfaces. The first step in this process is to find, up to any desired level of accuracy, the true intersecting curve.

The surface/surface intersection problem at the core of the `Junction` is due to Barnhill and Kersey.[31] Let the extracted patches pertaining to the `Wing` and `Fuselage` objects be denoted by $\mathbf{S}_0(u,v)$ and $\mathbf{S}_1(s,t)$, respectively. Given a starting point $\mathbf{P}_0$, whose projected parameter values (u_0, v_0, s_0, t_0) satisfy $\mathbf{S}_0(u_0,v_0) \cong \mathbf{S}_1(s_0,t_0)$, it is marched along the intersection curve until a boundary is reached. Each marched point follows a two-step process. The first step consists of determining a local, unit step direction $\mathbf{V}$ and step length L to guess a new approximation point $\widehat{\mathbf{P}}_1 = \mathbf{P}_0 + L\,\mathbf{V}$. Initially, $\mathbf{V}$ is determined by intersecting the tangent planes of the two surfaces at $\mathbf{P}_0$; all remaining direction vectors are taken as the difference of previous intersection points (backtracking). As for L, it is obtained from an adaptive method that is based on curvature ρ and an angular tolerance ANT:

$$L = \min\left(\rho \times \mathrm{ANT},\ \mathrm{CRT}\right)$$

where CRT denotes a Curve Refinement Tolerance. Once $\widehat{\mathbf{P}}_1$ has been calculated, it is "relaxed" onto the true intersection curve by solving the following three nonlinear equations in four unknowns:

$$\mathbf{S}_1(s,t) - \mathbf{S}_0(u,v) = \mathbf{0}.$$

Given a first iterate $\left(\hat{u}_1, \hat{v}_1, \hat{s}_1, \hat{t}_1\right)$, the above underdetermined system can be linearized and interpreted geometrically as the intersection between the tangent plane to $\mathbf{S}_0$ at $(\hat{u}_1, \hat{v}_1)$ and the tangent plane to $\mathbf{S}_1$ at $\left(\hat{s}_1, \hat{t}_1\right)$. The midpoint of that intersection is chosen as the next iterate. Convergence is reached when $\|\mathbf{S}_1(s,t) - \mathbf{S}_0(u,v)\| \leq$ Same Point Tolerance (SPT), where then $\mathbf{P}_1$ becomes the next $\mathbf{P}_0$. Assuming a root chord of 1 unit, typical values for ANT, CRT and SPT are on the order of $\mathrm{O}(10^{-2})$, $\mathrm{O}(10^{-3})$ and $\mathrm{O}(10^{-7})$, respectively.

Once an intersection curve has been approximated, there still remains the task of cutting and rearranging the surface topology of $\mathbf{S}_0(u,v)$ and $\mathbf{S}_1(s,t)$. For $\mathbf{S}_0$ this is straightforward, as the visible part is simply split at a point falling onto the `Wing`'s leading edge. In the case of $\mathbf{S}_1$, the process is a bit more involved; it is carried out in *parameter* space, where a designer is given the freedom to manipulate cubic Bézier curves that correspond, in design space, to the boundary curves of the ensuing surface network. This technique

[a]Trimmed surfaces are still viable when the geometry at hand is built for meshing purposes only, since to the best of our knowledge mesh generators such ICEM CFD can usually handle small gaps and holes.

allows for virtually any surface rearrangement of $\mathbf{S}_1$, but one that is adequate for an H-H gridding strategy is usually sought (see Figure 1(d)). Also note that by virtue of the reconstruction technique the resulting bicubic surfaces are guaranteed to be watertight, i.e. adjacent sides share the same control points and knot vectors.

D. Miscellaneous

GENAIR supplies a few other components aside from the main ones just described. The intent remains unchanged; it is merely to offer convenient classes that build directly on top of the NURBS library and automate most of the work, leaving the aircraft designer with what matters the most: creativity. For example, to create a `Nacelle` a designer needs only provide its radius, an `Airfoil`, and an optional angle at which the inlet may be tilted. Under the hood, the `Airfoil`'s characterizing curve $\mathbf{A}(u)$ is revolved around a central axis,[32] after which, if required, rows of control points are scaled against an imaginary plane rotated by the prescribed angle. Note the nine-point exact circle NURBS representation in Figure 1(e).

Basic structural components such as wing boxes and ribs are supported as well. Inputs are the number of ribs and the relative positioning of the front and rear spars. Relevant curves from the `Wing` are first extracted then linearly interpolated: e.g. let $\mathbf{C}_{\text{lo}}(u)$ and $\mathbf{C}_{\text{up}}(u)$ be respectively the lower and upper curves running parallel to the leading edge of a `Wing`; then for a fixed $\bar{u}$, $\mathbf{S}(\bar{u}, v)$ is a straight line segment connecting the points $\mathbf{C}_{\text{lo}}(\bar{u})$ and $\mathbf{C}_{\text{up}}(\bar{u})$. The resulting "ruled" surfaces,[25] see Figure 1(f), can be readily discretized with finite shell elements.

V. Application to Aerodynamic Shape Optimization

Once the initial upfront effort required by the generation of a new aircraft geometry has been invested, a shape optimization package must have the means to efficiently deform that same geometry (as opposed to redrawing it from scratch) upon changes in design parameters. Ideally, to do that the calling application would query GENAIR, which would not only regenerate the affected components automatically, but would also compute the associated surface sensitivities required by the next design iteration. Unfortunately, even though GENAIR has been built from the ground up to offer such capabilities, those are not functional at the time of writing this document. For now we employ an external deformer which is at any rate adequate to demonstrate the suitability of the geometries produced by GENAIR to high-fidelity ASO. Before presenting this technique applied to the lift-constrained drag minimization of a conventional regional jet, we first discuss the optimization framework responsible for updating the design variables in the first place.

A. Gradient-based Optimization Suite

The aerodynamic optimizer employed in this study can be broken down into four conceptually different but closely interlaced components: geometry parameterization, mesh movement, function evaluation and gradient evaluation. A two-level free-form deformation approach[18] links the first two, while the rest is handled by an adjoint-based algorithm.[19] Only a brief description of these tools is presented here; the reader is encouraged to consult the cited literature for more details.

Let $\mathbf{A}$ denote an xyz-coordinate block-column vector corresponding to all the NURBS control points that define a geometry. Then the first level of the two-level free-form deformation scheme involves embedding some or all of those control points inside one or more FFD volumes. Since these FFD volumes are in fact trivariate NURBS volumes, the geometry can then be modified by perturbing either individual or groups of control points pertaining to their lattices $\mathbf{B}$, a task normally performed by the optimizer. Note that by embedding control points of the underlying surfaces, rather than the usual grid points that lie on them, the exact geometry representation is never lost. At the second level, each block in the three-dimensional structured domain surrounding the geometry is assigned a NURBS volume whose global control net acts as a coarse grid $\mathbf{b} \supset \mathbf{A}$ (in words, there is a one-to-one correspondence between those control points in $\mathbf{b}$ embracing the geometry with those in $\mathbf{A}$). Once again, these volumes can be regarded as FFD volumes where this time around the full set of volume grid points $\mathbf{x}$ are embedded inside them.

In response to boundary alterations $\Delta\mathbf{A}$ resulting from the first level of FFD volumes, the control points $\mathbf{b}$ of the second FFD level are updated according to the equations of linear elasticity:[33]

$$\boldsymbol{\mathcal{M}}\left(\mathbf{b}, \mathbf{b}^{(0)}\right) = \mathbf{0}, \tag{3}$$

where the superscript 0 refers to the state before deformation. After solving Eq. (3), the entire computational mesh $\mathbf{x}$ is regenerated algebraically based on the NURBS volumes. This last operation is almost instantaneous, and since there are typically one to three orders of magnitude fewer control points $\mathbf{b}$ than grid points $\mathbf{x}$, the resources spent to converge Eq. (3) with a preconditioned conjugate gradient solver are only a few percent of what a grid-point-based deformation would cost.[19]

Note that the elasticity-based model is only valid under a small-strain assumption. If shape changes are sufficiently large, the displacements of the surface control points are generally required to be broken up into m increments:[33]

$$\mathbf{A}^{(i)} = \frac{i}{m}\left(\mathbf{A}^{(m)} - \mathbf{A}^{(0)}\right) + \mathbf{A}^{(0)}, \quad i = 1, 2, \ldots, m,$$

implying as many mesh solves.

Once the geometry has deformed and the surrounding volume mesh has adapted to it, a function evaluation is performed. The discrete objective function $\mathcal{J}$ relies on the discrete solution of both the pressure and viscous fields surrounding the aerodynamic surface. In this work, we ignore viscosity effects by considering the Euler equations only. Once discretized in space with finite-difference SBP-SAT operators,[34] these read $\frac{d\mathbf{q}}{dt} + \boldsymbol{\mathcal{R}} = \mathbf{0}$, from which

$$\boldsymbol{\mathcal{R}}\left(\mathbf{q}, \mathbf{b}^{(m)}\right) = \mathbf{0} \tag{4}$$

at steady-state. The variable $\mathbf{q}$ is a block-column vector of the conservative flow variables, while the dependence on $\mathbf{b}^{(m)}$, which is kept constant throughout a flow solve, is through the grid metrics. The nonlinear residuals making up Eq. (4) are converged through Newton's method, where at each iteration a system of linear equations is solved inexactly by a parallel Schur-preconditioned Jacobian-free Krylov iterative method.[34]

Next, the sensitivities of $\mathcal{J}$ to a set of design variables $\mathbf{v}$ must be determined. The design variables considered in this work are $\mathbf{v} = [\mathbf{v}_{\text{geo}}^T, \alpha]^T$, where $\mathbf{v}_{\text{geo}}$ is the vector of geometric design variables taken as a subset or entirety of $\mathbf{B}$ (or, as explained in Subsection B, any variable that handles parts of it simultaneously), and α is the freestream angle of attack. Since the size of $\mathbf{v}$ in typical ASO problems of interest is on the order of $\mathrm{O}(10^2)$, a logical choice for evaluating $d\mathcal{J}/d\mathbf{v}$ is the adjoint method.[35] As pointed out in Ref. 33, it is preferable to augment the traditional adjoint formulation with grid variables. In fact, an elegant formulation to the adjoint problem is:[19,33]

$$\begin{aligned}
\text{minimize} \quad & \mathcal{J}\left(\mathbf{v}, \mathbf{q}, \mathbf{b}^{(m)}\right), \\
\text{w.r.t.} \quad & \mathbf{v}, \mathbf{q}, \mathbf{b}^{(m)}, \\
\text{s.t.} \quad & \boldsymbol{\mathcal{M}}^{(i)}\left(\mathbf{A}^{(i)}(\mathbf{v}), \mathbf{b}^{(i)}, \mathbf{b}^{(i-1)}\right) = \mathbf{0}, \qquad i = 1, 2, \ldots, m, \\
& \boldsymbol{\mathcal{R}}\left(\mathbf{v}, \mathbf{q}, \mathbf{b}^{(m)}\right) = \mathbf{0}.
\end{aligned}$$

Typical of such a constrained optimization problem, a Lagrangian,

$$\begin{aligned}
\mathcal{L} &= \mathcal{L}\left(\boldsymbol{\lambda}^{(i)}, \boldsymbol{\psi}, \mathbf{q}, \mathbf{b}^{(i)}, \mathbf{v}\right), \qquad i = 1, 2, \ldots, m, \\
&= \mathcal{J}\left(\mathbf{v}, \mathbf{q}, \mathbf{b}^{(m)}\right) + \sum_{i=1}^{m} \boldsymbol{\lambda}^{(i)T} \boldsymbol{\mathcal{M}}^{(i)}\left(\mathbf{A}^{(i)}(\mathbf{v}), \mathbf{b}^{(i)}, \mathbf{b}^{(i-1)}\right) + \boldsymbol{\psi}^T \boldsymbol{\mathcal{R}}\left(\mathbf{v}, \mathbf{q}, \mathbf{b}^{(m)}\right),
\end{aligned}$$

is introduced. At optimality this Lagrangian must at least obey the first-order Karush-Kuhn-Tucker[36] conditions. The first two, $\partial\mathcal{L}/\partial\boldsymbol{\lambda}^{(i)} = \mathbf{0}$, $i = 1, 2, \ldots, m$, and $\partial\mathcal{L}/\partial\boldsymbol{\psi} = \mathbf{0}$, are satisfied provided Eqs. (3) and (4) are solved sequentially. The following two conditions, $\partial\mathcal{L}/\partial\mathbf{q} = \mathbf{0}$ and $\partial\mathcal{L}/\partial\mathbf{b}^{(i)} = \mathbf{0}$, $i = 1, 2, \ldots, m$, respectively lead to the equations for the Lagrange multipliers $\boldsymbol{\psi}$ and $\{\boldsymbol{\lambda}^{(i)}\}_{i=1}^{m}$, also known as the adjoint variables. These must be solved in reverse order relative to the ordering used to calculate $\mathcal{J}$:

$$\begin{aligned}
\left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \mathbf{q}}\right)^T \boldsymbol{\psi} &= -\left(\frac{\partial \mathcal{J}}{\partial \mathbf{q}}\right)^T, \\
\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(m)}}{\partial \mathbf{b}^{(m)}}\right)^T \boldsymbol{\lambda}^{(m)} &= -\left(\frac{\partial \mathcal{J}}{\partial \mathbf{b}^{(m)}}\right)^T - \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \mathbf{b}^{(m)}}\right)^T \boldsymbol{\psi}, \\
\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(i)}}{\partial \mathbf{b}^{(i)}}\right)^T \boldsymbol{\lambda}^{(i)} &= -\left(\frac{\partial \boldsymbol{\mathcal{M}}^{(i+1)}}{\partial \mathbf{b}^{(i)}}\right)^T \boldsymbol{\lambda}^{(i+1)}, \qquad i = m-1, m-2, \ldots, 1.
\end{aligned} \tag{5}$$

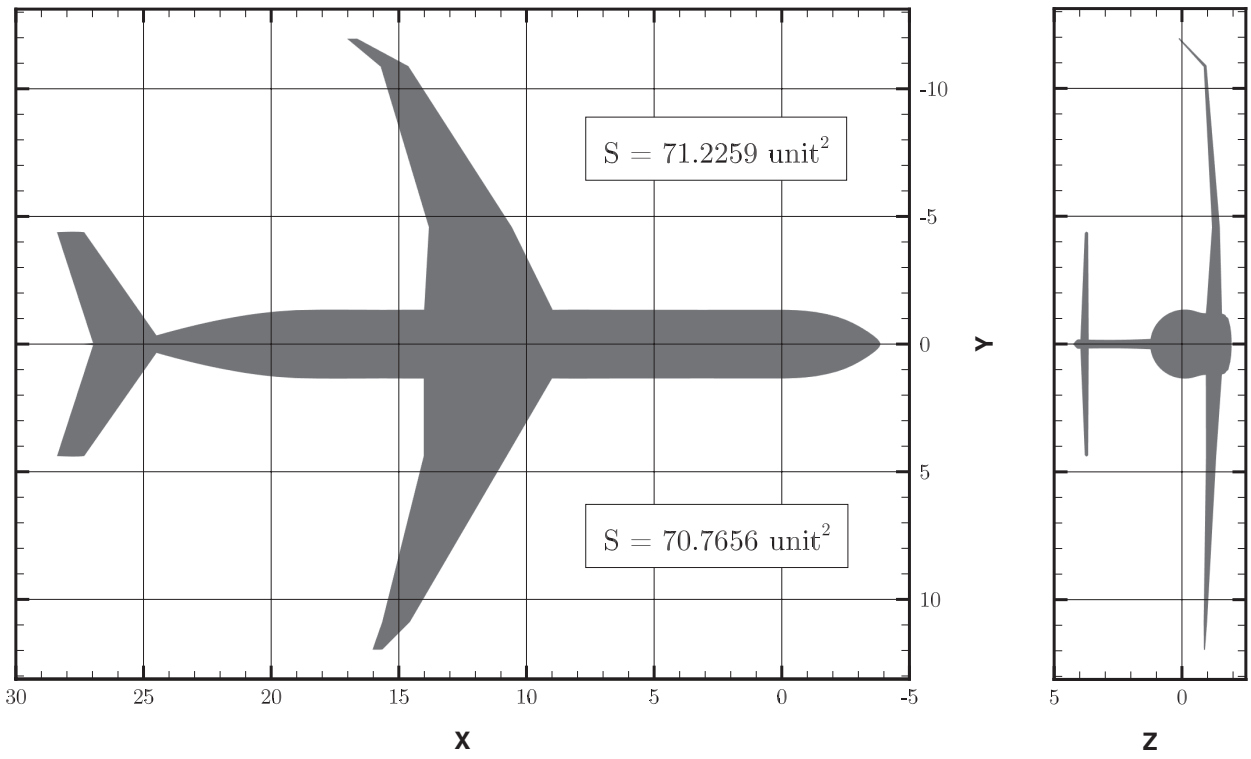


Figure 3. Planform (left) and frontal (right) views of the initial (bottom) and optimized (top) geometries. The projected area S increases slightly as a result of the optimization.

Finally, assuming all of the first four conditions are solved to a small enough tolerance, it follows that $\mathcal{L} = \mathcal{J}$, so the final condition, $\partial\mathcal{L}/\partial\mathbf{v} = \mathbf{0}$, is the total derivative of $\mathcal{J}$ with respect to $\mathbf{v}$, i.e. it is the desired gradient:

$$\frac{d\mathcal{J}}{d\mathbf{v}} = \frac{\partial\mathcal{J}}{\partial\mathbf{v}} + \sum_{i=1}^{m}\left(\boldsymbol{\lambda}^{(i)T}\frac{\partial\boldsymbol{\mathcal{M}}^{(i)}}{\partial\mathbf{A}^{(i)}}\frac{\partial\mathbf{A}^{(i)}}{\partial\mathbf{A}^{(m)}}\frac{\partial\mathbf{A}^{(m)}}{\partial\mathbf{v}}\right) + \boldsymbol{\psi}^T\frac{\partial\boldsymbol{\mathcal{R}}}{\partial\mathbf{v}}. \tag{6}$$

Once computed it is the task of an optimizer to choose an appropriate step direction and length that shall minimize $\mathcal{J}$. For that purpose, we use the optimization package SNOPT,[37] capable of handling problems with large numbers of design variables. SNOPT is based on the sequential-quadratic-programming paradigm and approximates the Hessian of its own Lagrangian using the quasi-Newton method of Broyden, Fletcher, Goldfarb and Shanno.[36]

B. Drag Minimization of a Conventional Regional Jet

We now present an optimization performed on a conventional aircraft generated by GENAIR. The aircraft is similar in features to the Bombardier CRJ-700,[38] with the main differences being that the nacelle is not included in the analysis and the winglet is originally not canted. The main wing is cranked and oriented at 0 degrees relative to the fuselage. Its cross-sections are based on the RAE 2822 transonic airfoil, except for the winglet's tip, which utilizes the NACA 0012 series airfoil. The CFD grid around the entire aircraft consists of 258 blocks and totalizes $2,623,600$ nodes, whereas the control grid has the same number of blocks but only $208,053$ nodes. The freestream Mach number is fixed at 0.79 with an initial angle of attack of 2 degrees. Each block is partitioned to one CPU, yielding typical wallclock convergence times of 61 s and 532 s for one mesh movement increment and one function evaluation, respectively.

The objective of the simulation is to minimize the sum of induced and wave drag components with respect to α as well as changes in the aerodynamic shape of the main wing. To control this shape we define three FFD volumes (one for each wing section, including the winglet) appended one after the other and in which surface control points are embedded. Each FFD volume is a $12 \times 2 \times 2$ NURBS volume, cubic in the streamwise (X) direction and linear in both the spanwise (Y) and vertical (Z) directions. Instead of choosing

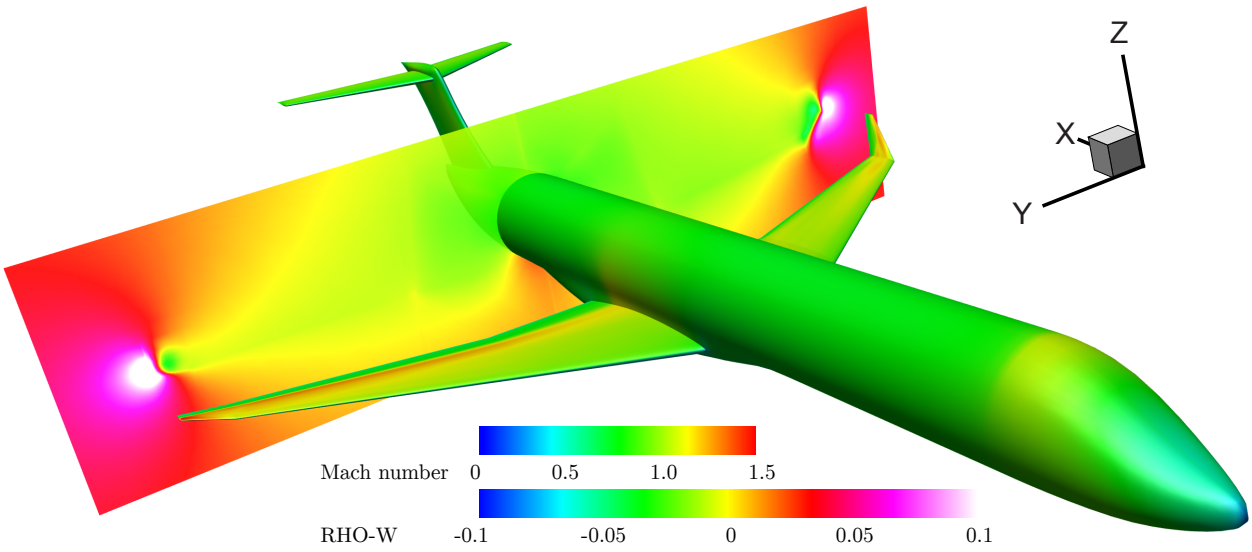


Figure 4. Initial (left) and optimized (right) aircraft shapes. The optimizer greatly reduced the strength of the shock located on the upper surface of the wing, as seen by the Mach contours, and has moved the winglet up to produce a nonplanar wake structure, as seen by the contours of the vertical component of momentum.

each coordinate of every FFD lattice control point as design variables (of which, accounting for overlapping control points, there are 288), we group together all the control points lying on the same spanwise station in such a way that the effective design variables become the sweep, dihedral, and span of any given wing section. We also give each control point the freedom to scale vertically in its own cross-sectional plane in order to allow different airfoil profiles to develop. Summarizing, in addition to α there are 7 global and 72 scaling design variables, for a total of 80 design variables.

Such design variables are appropriate to explore a wide variety of planforms; however due to the lack of a structural model we put bounds on the extent of the possible deformations. In particular, overall span is not allowed to increase, nor is the span of the winglet. It is also impossible for the winglet's dihedral to grow negative, but its tip is free to translate from its original position by as much as 1 unit in the streamwise, vertical and negative spanwise directions. Similarly, the crank between the first and second wing sections can move as far as 0.2 unit in all directions. In order to maintain a realistic design we restrict the control points lying in any given cross section to scale by no less than 0.7 and no more than 1.3 based on a reference point lying on the cross section's midline, and we also impose an equality constraint on the lift such that it is met exactly at the start of the optimization. One more adjoint solve is required for this last constraint due to its nonlinearity, but fortunately it is available only at a fraction of the cost of a function evaluation. Finally, the angle of attack is free to vary by ± 2 degrees.

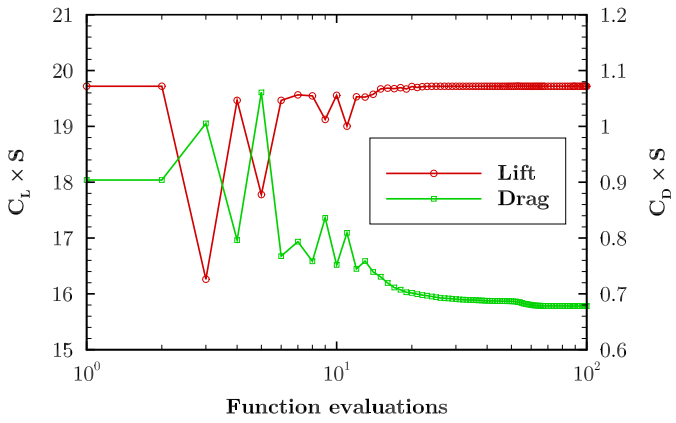


Figure 5. Convergence histories of the lift-constrained drag minimization problem. We report L/q_∞ and D/q_∞ as opposed to C_L and C_D, since the projected area S varies throughout the optimization.

The lift and drag values of the full configuration plotted against the number of function evaluations as the optimization progressed can be found in Figure 5 on the left. The optimality measure reported by SNOPT was reduced by 2 orders after 100 function evaluations at which point we terminated the simulation. The problem was thus not fully converged, yet the drag normalized by the dynamic pressure decreased from 0.904 unit^2 to 0.660 unit^2 (a 27% reduction) while satisfying the lift constraint to 7 significant digits. Based on the initial and final projected areas of 70.8 unit^2 and 71.2 unit^2,

respectively, this corresponds to respective coefficients of 0.279 and 0.277 for lift, and of 0.0128 and 0.00926 for drag. The final angle of attack is 1.66 degrees. The winglet was swept back and sheared up against the maximum allowable bounds as can be seen in Figure 3, an expected result given that lift and span were both constrained. The optimizer also successfully weakened the shock by a fair amount, see Figure 4, although it could not entirely eliminate it due to the lack of geometric flexibility or incomplete convergence. While more FFD control points could potentially be added along the span to remedy this situation, it would remain difficult to control the region immediately following the wing's root without perturbing the fuselage. This fact further goes to demonstrate the importance of integrating a geometry control system such as GENAIR within the shape optimization loop, a task that we intend on pursuing in the future.

The careful reader will have noticed that, contrary to common wisdom, the sweep of the final wing has not increased, even though the optimizer had such control. We would like to point out that the optimizer did increase it in early stages of the optimization, but eventually stepped back to favor a maximum relative sweep between the wing and winglet. Whether or not this mechanism leads to an actual local optimum is unclear since the optimizer did not reach deep convergence. In any case we would like to reiterate that this problem as a whole does not represent a practical optimization example (i.e. one that considers structures, off-design performance, mission analysis, etc.) but rather one that simply demonstrates the use of a baseline geometry drawn with GENAIR in the context of high-fidelity ASO.

VI. Conclusions and Future Work

A new tool, named GENAIR, specializing in the generation of unconventional aircraft geometries has been presented. It relies on a component-based approach where any number of `Fuselage`, `Wing`, `Junction`, etc. objects can be easily and arbitrarily assembled through an interactive *IPython*/OpenGL user interface. The flexibility of this tool was demonstrated through the design of several baseline unconventional aircraft, including a joined and truss-braced wing concept. Initial designs each require a few person-hours, but, once completed, any of the components can be automatically regenerated in a matter of seconds or less. This, combined with the fact that NURBS surfaces making up the final geometries are both watertight (untrimmed) and topologically four-sided (nondegenerate), will enable component-based ASO and MDO in the future. Presently, we conduct gradient-based, high-fidelity ASO by employing a general two-level FFD scheme that successively morphs both the surface and volume CFD grids while retaining analytical surface representations. Using this technique we demonstrated the applicability of a regional jet produced by GENAIR to a high-fidelity ASO problem by reducing the sum of its induced and wave drag components by as much as 27% while maintaining its initial lift.

Acknowledgments

We are thankful for the financial support provided by the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT), the Natural Sciences and Engineering Research Council of Canada (NSERC), the Ontario Graduate Scholarship (OGS), and the University of Toronto. Computations were performed on the gpc supercomputer at the SciNet HPC Consortium.

References

[1]Kroo, I. M., "Nonplanar wing concepts for increased aircraft efficiency," *VKI Lecture Series on Innovation Configurations and Advanced Concepts for Future Civil Aircraft*, Rhode-St-Genese, Belgium, June 2005.

[2]Liebeck, R. H., "Design of the Blended Wing Body Subsonic Transport," *Journal of Aircraft*, Vol. 41, No. 1, January-February 2004, pp. 10–25.

[3]Claus, R., Follen, G. J., and Haimes, R., "CAPRI: Computational Analysis PRogramming Interface. 'A CAD infrastructure for Aerospace Analysis and Design Simulations'," Tech. rep., National Aeronautics and Space Administration, Cleveland, Ohio, 2000.

[4]Fudge, D., Zingg, D. W., and Haimes, R., "A CAD-Free and a CAD-Based Geometry Control System for Aerodynamic Shape Optimization," *43rd AIAA Aerospace Sciences Meeting,* AIAA Paper 2005-0451, Reno, Nevada, January 2005.

[5]Jones, W. T., Lazzara, D., and Haimes, R., "Evolution of Geometric Sensitivity Derivatives from Computer Aided Design Models," *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference,* AIAA Paper 2010-9128, Fort Worth, Texas, September 2010.

[6]Hahn, A. S., "Vehicle Sketch Pad: Parametric Geometry for Conceptual Aircraft Design," *48th AIAA Aerospace Sciences Meeting,* AIAA Paper 2010-0657, Orlando, Florida, January 2010.

[7]Tomac, M. and Eller, D., "From geometry to CFD grids — An automated approach for conceptual design," *Progress in Aerospace Sciences*, Vol. 47, No. 8, November 2011, pp. 589–596.

[8]Suwaratana, D. L. and Rodriguez, D. L., "A More Efficient Conceptual Design Process Using the RAGE Geometry Modeler," *49th AIAA Aerospace Sciences Meeting,* AIAA Paper 2012-0159, Orlando, Florida, January 2011, [http://www.desktop.aero/rage.php, accessed 07/11/12].

[9]Hwang, J. T. and Martins, J. R. R. A., "GeoMACH: Geometry-Centric MDAO of Aircraft Configurations with High Fidelity," *14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference,* AIAA Paper 2012-5605, Indianapolis, Indiana, September 2012.

[10]Cavallaro, R. and Frediani, A., "A Code for Shape Generation and Aerodynamic Design of Aircraft," *Variational Analysis and Aerospace Engineering: Mathematical Challenges for Aerospace Design*, edited by G. Buttazzo and A. Frediani, Optimization and Its Applications, Springer, 2012, pp. 117–139.

[11]Prochnow, S., Stenz, G., and Ziemer, S., "On development environments for aircraft modeling," *IEEE Aerospace Conference*, Big Sky, Montana, March 2012.

[12]"AVID PAGE (Parametric Aircraft Geometry Engine)," [http://www.avidaerospace.com/software/avid-page, accessed 26/05/13].

[13]"Pacelab APD," [http://www.pace.de/products/preliminary-design/pacelab-apd.html, accessed 26/05/13].

[14]Heath, C. and Gray, J., "OpenMDAO: Framework for Flexible Multidisciplinary Design, Analysis and Optimization Methods," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference,* AIAA Paper 2012-1673, Honolulu, Hawaii, April 2012.

[15]Perez, F. and Granger, B. E., "IPython: a system for interactive scientific computing," *Computing in Science & Engineering*, Vol. 9, No. 3, May-June 2007, pp. 21–29, [http://www.ipython.org, accessed 07/11/12].

[16]Piegl, L. A., "On NURBS: a survey," *Computer Graphics and Applications*, Vol. 11, No. 1, 1991, pp. 55–71.

[17]Blake, M. W., Kerr, P. A., Thorp, S. A., and Chou, J. J., "NASA Geometry Data Exchange Specification for Computational Fluid Dynamics (NASA IGES)," Tech. Rep. 1338, National Aeronautics and Space Administration, 1994.

[18]Gagnon, H. and Zingg, D. W., "Two-Level Free-Form Deformation for High-Fidelity Aerodynamic Shape Optimization," *14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference,* AIAA Paper 2012-5447, Indianapolis, Indiana, September 2012.

[19]Hicken, J. E. and Zingg, D. W., "Aerodynamic Optimization Algorithm with Integrated Geometry Parameterization and Mesh Movement," *AIAA Journal*, Vol. 48, No. 2, February 2010, pp. 400–413.

[20]Oliphant, T. E., "Python for Scientific Computing," *Computing in Science & Engineering*, Vol. 9, No. 3, May-June 2007, pp. 10–20, [http://www.numpy.org., accessed 26/05/13].

[21]Holkner, A., "pyglet: a cross-platform windowing and multimedia library for Python," [http://www.pyglet.org, accessed 26/05/13].

[22]Fletcher, M. C., "PyOpenGL 3.x - The Python OpenGL Binding," [http://www.pyopengl.sourceforge.net, accessed 26/05/13].

[23]IGES/PDES Organization, "Initial Graphics Exchange Specification (IGES)," Version 5.3, 1996.

[24]OPEN CASCADE S.A.S., "Open CASCADE Technology, 3D modeling & numerical simulation," [http://www.opencascade.org, accessed 07/11/12].

[25]Piegl, L. and Tiller, W., *The NURBS Book*, Springer, 2nd ed., 1997.

[26]Nemec, M. and Aftosmis, M. J., "Parallel Adjoint Framework for Aerodynamic Shape Optimization of Component-Based Geometry," *49th AIAA Aerospace Sciences Meeting,* AIAA Paper 2011-1249, Orlando, Florida, January 2011.

[27]Sederberg, T. W. and Parry, S. R., "Free-Form Deformation of Solid Geometric Models," *SIGGRAPH 86*, Dallas, Texas, August 1986.

[28]Shetty, S. and White, P. R., "Curvature-continuous extensions for rational B-spline curves and surfaces," *Computer-Aided Design*, Vol. 23, No. 7, September 1991, pp. 484–491.

[29]Gordon, W. J., "Spline-blended surface interpolation through curve networks," *Journal of Mathematics and Mechanics*, Vol. 18, No. 10, 1969, pp. 931–952.

[30]Sederberg, T. W., Finnigan, G. T., Li, X., Lin, H., and Ipson, H., "Watertight trimmed NURBS," *ACM Transactions on Graphics*, Vol. 27, No. 3, 2008, pp. 1–8.

[31]Barnhill, R. E. and Kersey, S. N., "A marching method for parametric surface/surface intersection," *Computer Aided Geometric Design*, Vol. 7, No. 1–4, June 1990, pp. 257–280.

[32]Piegl, L. and Tiller, W., "Approximating Surfaces of Revolution by Nonrational B-Splines," *IEEE Computer Graphics and Applications*, Vol. 23, No. 3, 2003, pp. 46–52.

[33]Truong, A. H., Oldfield, C. A., and Zingg, D. W., "Mesh Movement for a Discrete-Adjoint Newton-Krylov Algorithm for Aerodynamic Optimization," *AIAA Journal*, Vol. 46, No. 7, July 2008, pp. 1695–1704.

[34]Hicken, J. E. and Zingg, D. W., "Parallel Newton-Krylov Solver for the Euler Equations Discretized Using Simultaneous-Approximation Terms," *AIAA Journal*, Vol. 46, No. 11, November 2008, pp. 2273–2786.

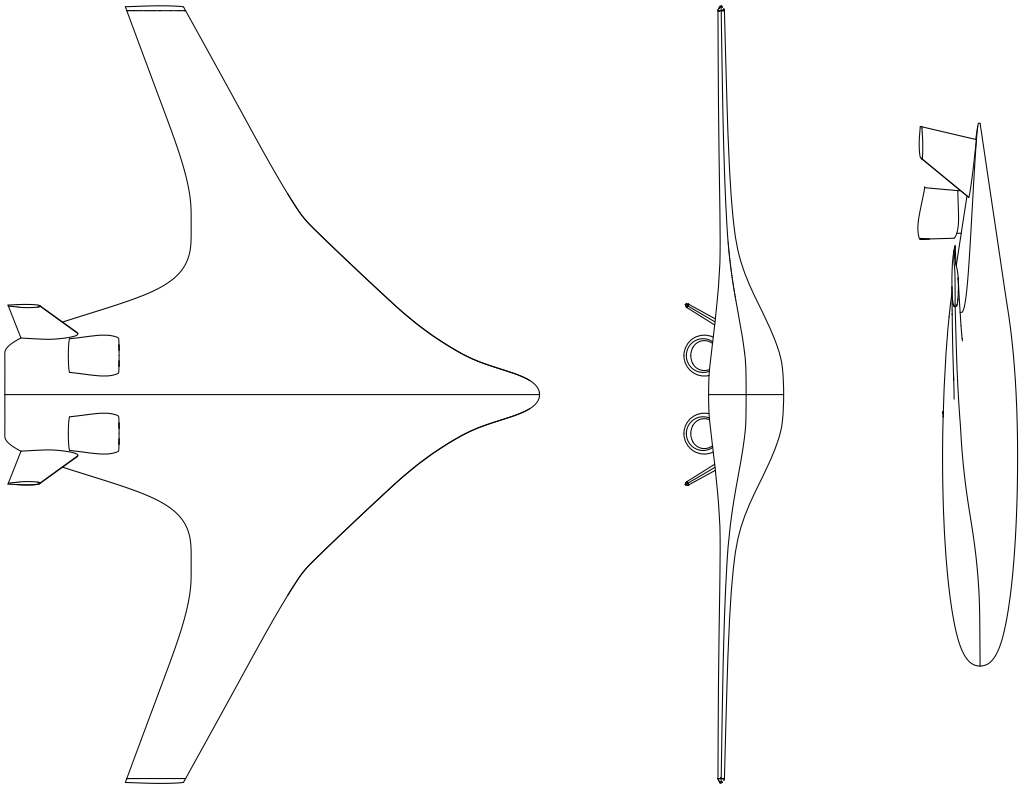
[35]Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, No. 3, September 1988, pp. 233–260.

[36]Nocedal, J. and Wright, S. J., *Numerical Optimization*, Springer, 2nd ed., 2006.

[37]Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Review*, Vol. 47, No. 1, 2005, pp. 99–131.

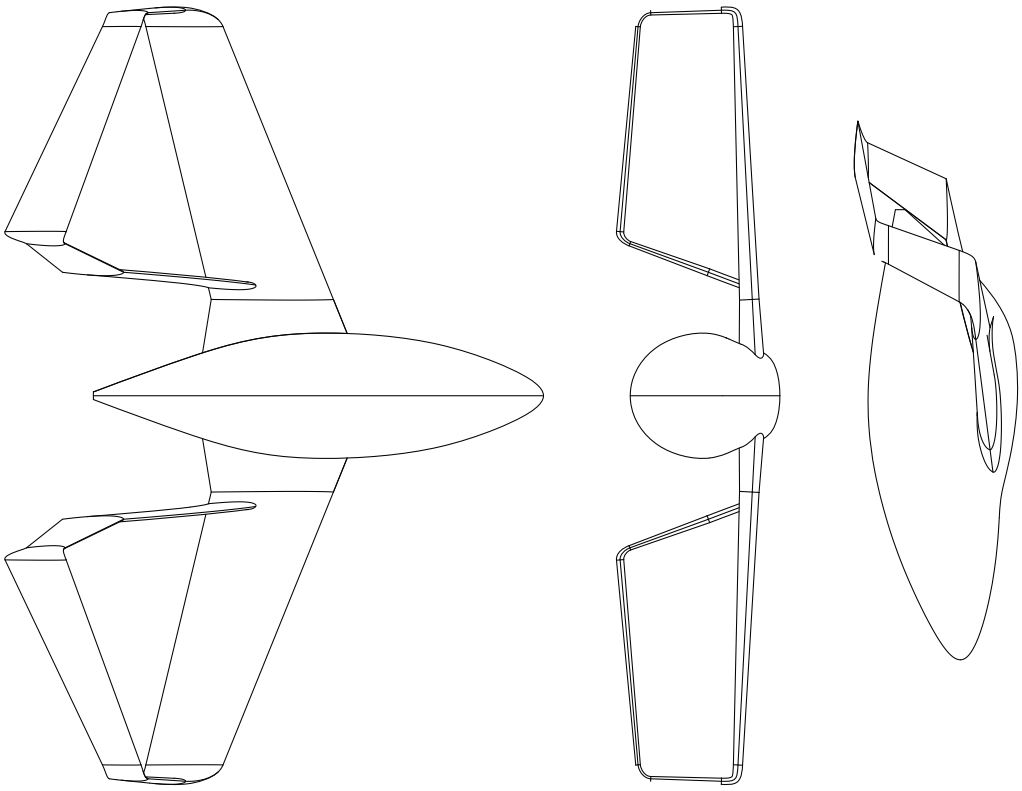
[38]Kafyeke, F., Pépin, F., and Kho, C., "Development of High-Lift Systems for the Bombardier CRJ-700," *23rd Congress of International Council of the Aeronautical Sciences*, Toronto, Ontario, September 2002.

A. 3-views of the Unconventional Aircraft Found in Figure 2



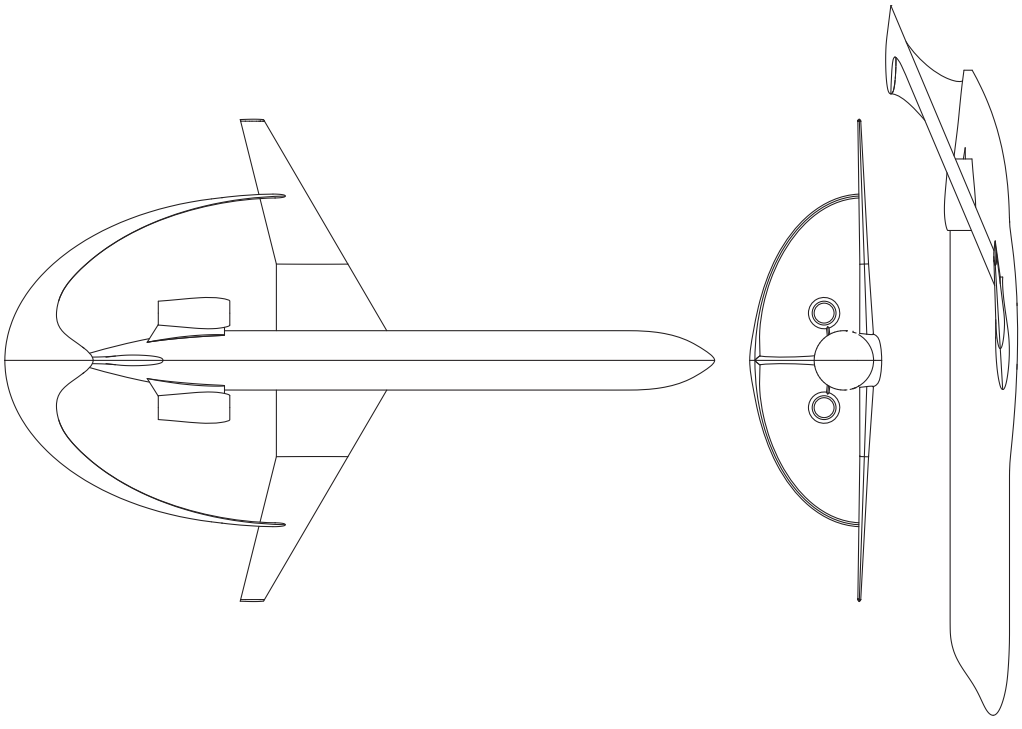
Number of wing objects	6	Wingspan	36.37 m	1 m
Airfoils used	WHITCOMB NACA SC-0012	Overall length	25.46 m	
		Overall height	4.77 m	
		Wetted area	773.01 m^2	

Figure 6. Blended-wing-body regional transport.



Number of wing objects	18	Wingspan	12.10 m	1 m
Airfoils used	NLF 0416	Overall length	8.36 m	
	NACA 0012	Overall height	2.55 m	
		Wetted area	92.30 m^2	

Figure 7. Synergy personal aircraft.



Number of wing objects	9	Wingspan	21.92 m	1 m
Airfoils used	RAE 2822	Overall length	32.10 m	
	DAVIS B-24	Overall height	5.93 m	
	NACA SC-0012	Wetted area	489.79 m^2	

Figure 8. Joined-wing regional jet.

Number of wing objects	13	Wingspan	19.98 m
Airfoils used	RAE 2822	Overall length	19.92 m
	DAVIS B-24	Overall height	5.15 m
	NACA 0012	Wetted area	305.93 m^2

$\frac{1}{m}$

Figure 9. C-wing business jet.

Number of wing objects	14	Wingspan	49.64 m
Airfoils used	HSNLF 0213	Overall length	29.09 m
	SELIG 9026	Overall height	4.05 m
		Wetted area	817.98 m^2

$\frac{1}{m}$

Figure 10. Truss-braced-wing cargo aircraft.