On The Construction of Aircraft Conceptual Geometry for High-Fidelity Analysis and Design

Robert Haimes* and Mark Drela†

Aerospace Computational Design Laboratory

Massachusetts Institute of Technology, Cambridge, Massachusetts, 02139

This paper examines the desirability and the challenges of incorporating high-fidelity geometry definition into the Multidisciplinary Design, Analysis and Optimization (MDAO) process earlier than currently practiced. A major objective is the ability to enable geometry definition for low-fidelity as well as high-fidelity analyses, in order to support the entire MDAO process from conceptual to detail design in a seamless manner. Another objective is the ability to support different disciplines such as both structural and aerodynamic analyses from the same geometry definition. Finally, there are the goals of ease of use and support for automation to minimize unnecessary or repetitive human effort.

It is argued that Constructive Solid Geometry (CSG) is the natural foundation for attaining these goals. Two different current user-level approaches which employ CSG at low level are considered: 1) CAD systems and their "feature" based view of construction, and 2) Bottom-Up methods which generate solid "components". Although Bottom-Up methods do not have the turn-key features of commercial CAD systems, it is clear that their flexibility and potential open nature is an advantage in the long term, especially if geometric design-gradient information is required for optimization.

To realize the MDAO objectives via the Bottom-Up approach, a new software suite, the Electronic Geometry Aircraft Design System (EGADS), has been developed. It is a relatively simple and compact Open-Source Object-Based API built on top of the extensive OpenCASCADE solid-modeling kernel. EGADS routines implement relatively high-level operations which insulate the user from OpenCASCADE's size and complexity, and for maximum flexibility can be driven by either C, C++, or FORTRAN user applications. The basic features and constructs of EGADS are described, and an example application is presented to demonstrate its capabilities and effectiveness.

I. Introduction

A. Background

AIRCRAFT conceptual design traditionally performs mission analysis, sizing, and configuration down-select of candidate designs via empirical or low-fidelity physics analyses. The geometry parameters at this stage can, in practice, define the overall shape or even the outer mold-line (OML) of the aircraft to a degree sufficient for low-order aerodynamic analyses such as Vortex-Lattice or Panel Methods, or for simple structural analyses such as Simple Bending/Torsion Beam Theory. But they do not have the ability to generate completely realizable (3D and closed) geometry needed to fully define the aircraft or components for high-fidelity CFD analyses, FEA analyses, or rapid-prototyping systems. An archetypical example of this kind of Conceptual Design software is the Program for Aircraft Synthesis Studies (PASS).[1] Its modules include low-fidelity models for aerodynamics, structures, weights, propulsion, stability and noise evaluated

*Principal Research Engineer, Department of Aeronautics & Astronautics, AIAA Member.

†Terry J. Kohler Professor, Department of Aeronautics & Astronautics, AIAA Fellow.

over mission segments that include takeoff, climb, cruise, approach, and landing. PASS does include a set of high-level geometry design variables describing the wing planform, empennage planform, and general fuselage features, but these cannot generate watertight geometric models.

The OML is typically considered early in traditional aircraft design, and the details of other subsystems, in particular structural layouts, are not examined until later design phases. As reported by Jouannet,[2] the configuration's OML geometry design can be impacted by these subsystems, which may not be discovered until high-fidelity analysis is done in later development phases. This often results in engineering work-arounds which consume performance margins and increase cost, and hence should be avoided if possible.

Further complications tend to arise whenever the geometry definition is scattered over many different analysis methods, which can span different disciplines as well as different fidelity levels within a discipline. For example, a Vortex-Lattice method for initial aero analysis requires only surface planform outlines and airfoil camberlines, while a Panel method for intermediate aero analysis requires the overall OML. Simple beam analysis versus higher-order beam or FEA analyses likewise require different levels of geometry definition. The complication arises from the likelihood that geometry adjustments from redesign based on one method is likely to be inconsistent with the other geometry definitions, which ultimately must be resolved by human intervention in some typically ad-hoc manner.

B. Motivation for Early High-Fidelity Geometry

One solution to the various difficulties and complications introduced above is to form a 3D closed model of the aircraft earlier, even as early as the conceptual design phase, and interrogate this to generate geometry inputs for all analyses. As outlined by Lazzara[3] many advantages arise from such an early 3D model implementation. If a 3D model incorporates a low- and high-fidelity parameterization that is consistent with needed geometry requirements, then higher fidelity analysis becomes possible alongside low-fidelity tools. For conceptual design, this leads to a greater confidence in calculated performance metrics for configuration down-select. Also, there is no need to continuously create higher-level geometry versions as the design matures, because this high-level geometry already exists. Attempts at developing a framework that accomplishes this have been made by Amadori et al., but relatively few others.[4]

The use of a 3D model as a central geometry source is illustrated in Figure 1, where four geometric representations are shown to potentially satisfy low-fidelity mission analysis (abstract design description), low- and high-fidelity aerodynamics analysis (wireframe model and surface mesh), and high-fidelity structural analysis (surface mesh on structural components). In this light, the 3D model is at the center of a design framework (where it should be) and all analyses become consistent via this parent geometry.

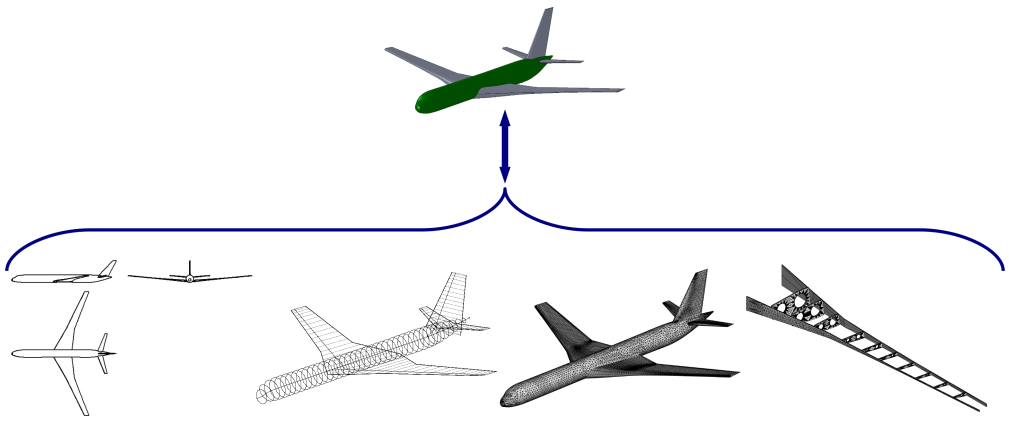


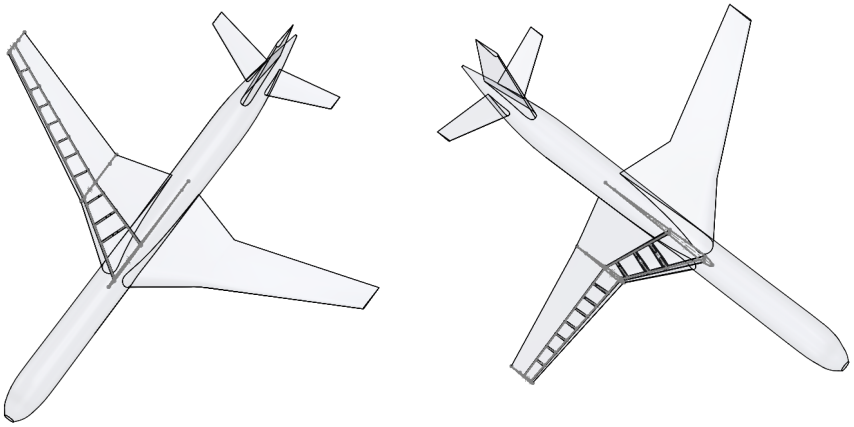
Figure 1. A fully-realizable, high-fidelity, 3D model geometry can be the source of different lower-fidelity geometric representations which are applicable for multidisciplinary and/or multi-fidelity analysis.

This concept of Multi-fidelity Geometry was introduced by Lazzara[3] in the context of producing various OML realizations, and was further extended by Lazzara et al.[5] to include structural layouts. The latter

extension is natural because the outer mold-line directly constrains the structural layout configurations. By generating structural components using construction methods employed on the OML, the same properties of malleability, robustness, and flexibility are expected to exist throughout the entire process.

Once the structural components are generated, a validation of proper design motion is required to ensure that design space bounds are observed. A baseline model test case which displays regeneration robustness is shown in Figure 2(a). The structural design layout consists of two spars, fore and aft, with ten evenly spaced ribs along the wing semi-span. The associated assembly datum references were made between the spar and wing surface, followed by references between the ribs and spars.

Regeneration of the baseline model is made after moving the wing tip forward, as seen in Figure 2(b). The previously aft-swept wing planform thus becomes a forward-swept wing planform. Successful regeneration occurs, where the spar design motion matched that of the leading/trailing edge guide curves, thereby also exhibiting a change in sweep angle; rib components also underwent the correct design motion and reach the expected final orientation. Since the spars and ribs reside within the wing design space, regeneration of the geometry associated with the structural layout is possible when representing a different region of the wing design space. Unsuccessful regeneration would occur if the structural layout design space were defined to extend beyond the design space bounds of the OML; evidence of such problems would appear as surface-crossings, intended design motions beyond datum reference limits, and other geometry configurations that break the assembly integrity.



(a) (b)

Figure 2. (a) A baseline configuration used in the regeneration robustness tests. (b) Results from an initial regeneration robustness test, showing the intended design motion for the structural layout after changing the wing from an aft-swept to forward-swept configuration (from Lazzara et al.[5]).

The rest of the paper discusses two approaches that can be used to realize this multidiscipline/multi-fidelity geometric handling where the 3D closed model plays a central role.

II. A CAD-based Approach

A. File Formats and Geometry Access

The usual connection between high-fidelity analysis codes and geometry is performed by a grid generator. The logical choice to transmit geometry to the grid generator is through standard file formats. The IGES file format contains data that is defined as disjoint and unconnected surfaces and curves, with no explicit notion of topology (the hierarchical connectivity of these entities). However, 3D meshing software ultimately requires such topological information to realize a closed "watertight" model. Much effort is therefore needed to take the IGES data, trim the curves and surfaces, and then deduce the topology. This process is particularly onerous when the source of the geometry is a CAD system or a Solids-based geometry kernel. In this case

the part's description in the kernel is probably closed with defined topology, so it's unfortunate that in the translation to IGES for file transmission this important topology information is needlessly lost.

The STEP file format supports topology as well as geometry. This is therefore the preferable file type to use for the transmission of closed models. Surprisingly, this format is seldom used in practice. This may be due to the fact that constructing a STEP reader is complex and requires a complete solid modeling geometry kernel to deal with the data. Also, transferring data via STEP is not without its own set of problems. Specifically, each CAD system or geometry kernel uses a different mathematical formulation to represent the same types of surfaces, and also have different tolerances for closure. After reading a solid part, one may find that the model is now open, again requiring some form of patching.

An alternative approach to geometric file transmission is to couple directly to the kernel, or the source of the geometry. A direct vendor neutral API allows an analysis builder to access the geometric data without programming directly for each system. CAPRI (Computational Analysis PRogramming Interface)[6] is an example of such an API which also provides a solution to the CAD dependency issue. Coupling to any supported CAD package or geometry kernel is both unified and simplified by using the CAPRI definition of geometry (with topology) where native access to the geometry and the topological data is granted through its API.

One clear advantage to this approach is that the geometry never needs to be translated and hence remains simpler and closed. Another advantage is that writing and maintaining the grid generator (coupled to the CAD system) can be done *once* through the API. All of the major CAD vendors are then automatically supported.

B. Boundary Representations

Boundary Representations (BReps) are the standard data model that holds both the geometric and topological entities that supports the concept of a "solid", as well as other non-manifold aggregations. See Table 1 for a fairly complete topological view. BReps have a tolerance that determines the meaning of "closure" for connected entities. This means that the Nodes that bound an Edge are probably not on the underlying curve. Specifically, Edges that bound a Face (through the Loops) do not necessarily sit on the supporting surface. However, for a valid closed solid all that is required is that the bounding objects (Nodes/Edges) be within a specified tolerance of the higher dimensioned entity (Edges/Faces). Therefore, for any precision higher than the tolerance, gaps and overlaps may exist in the geometry definition. This tolerance is generally much larger than those associated with double precision floating-point arithmetic.

To deal with gaps and overlaps without a program halt which then requires intervention, most BRep-based applications must "fix" the geometry. This usually entails translating the geometric definition to another simpler representation where the bounding entities fall closer to the higher-dimensional object. This type of translation has a variety of side effects, including:

- **Inconsistency**: Not querying the same geometry. Since the geometry has changed, the representation is different than in the source system.
- **Errors**: By changing the geometry, unquantified errors are introduced into the process.
- **Complexity**: At times additional Faces are required to close the model. There is no way to predict how many of these "sliver faces" may need to be introduced; moreover, slivers can cause significant problems for grid generators.
- **Not automatic**: There are always situations that cannot be healed in a *hands-off* manner. The requirement of user intervention is problematic for any fully automated process such as design optimization.

CAPRI's perspective is that the geometry in the CAD system is *truth* and should not be modified (though CAPRI may modify the topology). Therefore fixing (or "healing") the CAD's model is no longer a required part of the analysis procedure.

C. An Associative Triangulation

An API that only gives the programmer access to the BRep is a fairly difficult starting point for 3D meshing. The burden of deciphering the data and attempting to generate a discrete representation of the surfaces required for mesh generation is quite high. Fortunately, many grid generation systems used in CFD and other disciplines can use *Stereo Lithography* (STL) files as input. Combining a discretized view of the BRep as well as its geometry and topology can provide a complete, and easier to use, access point. A tessellation of the object that contains not only the mesh coordinates and supporting triangle indices but other data, such as the underlying surface parameters for each point, as well as the connectivity of the triangles, assists in traversing through and dissecting a complex part.

Although CAPRI's tessellations could be used as the starting point for computational analysis, that is not their intent. CAPRI sees only geometry, and it cannot anticipate the smoothness, resolution, or other requirements of the downstream applications. The triangulations approximate the geometry only, and some processing of the tessellation is expected in order to refine the triangulation to a state suitable for the physical problem being investigated. The triangulation can be enhanced through either physical or parameter space manipulation, using point "snap" and (u, v) surface evaluation routines provided by the CAPRI API.[7] The triangulation technique used within CAPRI displays the following characteristics:[8,9]

- **Robust**. It is imperative that the scheme work for all possible topologies and provide a tessellation that can be used.
- **Correct**. The triangulation is of no use if it is not true to the BRep model. The tessellation must be logically correct; i.e. provide a valid triangulation in the parameter space (u, v) of the individual surface. It must also be geometrically correct; i.e. depict a surface triangulation that truly approximates the geometry. This involves ensuring all facets have a consistent orientation with no creases or abrupt changes in triangle normals. Correctness in both physical and parameter space allows CAPRI based application enhancement schemes to operate in either or both.
- **Adjustable**. To minimize the post-processing of CAPRI's tessellation for a specific discipline or analysis, some a priori adjustment of the resultant quality is available. It must be noted however, that any criteria may not be met (especially near the bounds of an object in the BRep) due to issues of closure and modelling accuracy. This goal may conflict with the more important characteristic of being watertight and having a smooth surface representation.
- **No geometric translation**. To truly facilitate hands-off grid generation, anything that requires user intervention must be avoided. All data maintained within CAPRI is consistent with the BRep. An alternative or translated representation is not used, because then the result will be something different than resides within the source system.
- **Watertight**. Triangulated solids are closed and conformal, which allows for meshing without "fixing" geometry. For the tessellation of a BRep, this means that all Edge (trimming) curves terminate at consistent coordinates of the bounding Nodes and a single discretization for Edge curves be used on both surfaces sharing the common Edge. Each triangle side in the tessellation is shared by exactly two triangles, and the star of each vertex is surrounded and bounded by a single closed loop of sides. The triangulation is everywhere locally manifold. In a manifold triangulation, there are no voids, cracks or overlaps of any triangles that make up the solid.

D. Modifying Geometry for Design

Most all CAD systems support the Master-Model concept of representing an object (note that this is also referred to as "Parametric" or "Feature-based" CAD). The Master-Model describes the sequence of operations to build the geometry (usually) of a solid model. At its most basic level, it is an ordered list of extrude, revolve, merge, subtract and intersection operations. CAD systems support more meaningful abstractions, such as blends, lofts, fillets, drilled holes and bosses which encompasses *Constructive Solid Geometry* (CSG).

When the CAD model is regenerated, the operation list is interpreted by the CAD system to sequentially build the geometry of the part. This gives the operator the ability to construct a family of parts (or assemblies) by building a single instance. Many of the operations used in the construction can be controlled by parameters that may be adjusted. By changing these values, a new member of the family can be built by simply following the prescription outlined in the Master-Model definition.

The recipe may be simple, like a serial collection of primitive operations, but can also be complex, where operations are performed on previously or temporarily constructed geometry. The representation of this construction in most CAD systems is the form of a tree, usually referred to as the "Feature Tree". By supporting this method of construction, a direct API can provide both simple and powerful access to the CAD system. This approach is clearly outside the static view traditionally held of geometry. That is, this kind of access and control is not possible from any type of file transfer.

Within CAPRI, this tree is presented to the programmer in the form of "branches". Each of these entities has an index to identify where in the tree the reference is made. All indices are relative (that is they can occur anywhere in the tree – the assignment is usually given during initial parsing of the CAD internal structures). There is a special branch always given the index zero, the root of the tree. Therefore, the entire tree may be traversed starting at the root and moving toward the end of each branch. The branches terminate at leaves (branches that do not contain any children). To aid in traversing the tree toward the root the parent branch is always available. Unlike simple binary trees, a branch in CAPRI's Feature Tree may contain zero or more children.

Some branches may be marked "suppressible" – these features may be turned off, in a sense removing that branch (and any children of the branch) from the regeneration. This is powerful in that it allows for defeaturing the model, so that it may be made appropriate for the type of analysis at hand. For example: if fasteners are too small for a fluid flow calculation, they may be easily suppressed (if the Master-Model was constructed with this in mind). After part regeneration the resultant geometry would be simplified and the details associated with the fasteners would not be expressed.

Parameters are those components of the Master-Model that contain values (and should not be confused with the geometric parameterization). CAPRI exposes all parameters found in the model, but the programmer can only modify those that are adjustable (non-driven). This is a separate list from the Feature Tree, but references back to the associated branches where the values are used or defined. Parameters may be single or multi-valued and can be Booleans, integers, floating-points or strings.

This CAD perspective on parametric building of parts and assemblies is fine for driving the part using simple parameters but is problematic for shape design. For example, simple parameters may be used to define the planform of an aircraft, but are difficult to use to define the airfoil shape of the wing and tail components. The CAD operator would need to expose the curve/surface definition at a very fine and detailed level (i.e. knot and/or control points as the parameters) to allow for the exact specification of shapes. CAPRI avoids placing this burden on the CAD designer by exposing certain curves as multi-valued "parameters". These curves are obtained from independent sketched features in the model that later are used in solid generation as the basis for rotation, extrusion, blending and/or lofting. The curves can be modified, and when regenerated, the new part expresses the changed shape(s). This functionality is critical for shape design in general and specifically for aerodynamic shape design.

E. Parametric Sensitivities

The generation of design parameter sensitivity derivatives is required for gradient-based optimization. Typically, this is accomplished by differentiating the respective tools by hand, by using automatic differentiation tools, or by the use of complex variables. However, the geometric sensitivity derivatives are elusive when the geometry is defined by complex and often proprietary CAD software. With proprietary systems, software source code is unavailable and therefore differentiation of the source is not possible.

Other techniques have employed finite-differencing of distinct CAD model instances to define sensitivity derivatives. These methods are not only costly, but involve an assortment of challenges related to the proper definition of step size on finite precision computers. The cost is primarily associated with the potentially large number of instances (one for each geometric design variable) of the model that needs to be generated

for finite-differencing. A challenge is that the topology of each instance needs to remain the same. If a vertex appears or disappears in one instance then the change in its position from a previous instance is not defined.

An interesting observation is that the individual instance of a Master-Model is the subject of analysis, while it is the Master-Model together with its parametric sensitivities that are the subject of optimization.[10]

Each instance of the Master-Model results in a BRep. The BRep includes the model's topology which collects the geometric entities into their topological equivalent and provides the connectivity information. The topology is directly related to both the design intent of the Feature Tree and the construction methods of the underlying CAD system. Topological entities are driven explicitly by the design parameters or implicitly by construction operations. Therefore, analysis and optimization depends on information in the Master-Model parameter space as well as the BRep topology and gradient-based optimization requires the sensitivity of the model topology to the driving parameters.

To directly utilize the Master-Model within a gradient-based optimization context, one could imagine differentiating through the CAD system to directly obtain the geometric sensitivity to the parameters. However, this is not possible for reasons cited above. An alternative approach is presented by Jones[11] which makes direct use of the hierarchical associativity of the CAD features in a BRep to trace their evolution and thereby track sensitivity to design parameters.

F. Seamless Geometry Evolution Between Design Stages

The traditional design process starts from a conception stage where no actual geometry may be specified, to a final design where the part is fully realized down to the finest details. In a multidisciplinary design setting, one discipline may set some "parameters" before passing its information along to the next. Only when there is the requirement for more detailed analysis needing commensurate geometric properties will the design be *fleshed out*.

It should now be clear that if the design process changes from this traditional situation to one where the designer predefines the part's intent and possible expression (through a Master-Model definition) the following becomes feasible:

- **Consistency**. Each phase in the design process uses the same suite (or a subset of the suite) of parameters. Any parameter value change that produces a different geometry can be viewed by another stage in the process without writing and reading the geometry in files. The CAD part, regenerated with a particular set of parameter values and Feature Tree suppression states, uniquely describes the geometry.

- **Data Repository**. The CAD system and Product Data Management (PDM) and/or Product Lifetime Management (PLM) software can be used to track and maintain the design. Also, because the design is in the CAD system from the beginning, issues of manufacturability can be easily addressed early on and unrealistic expressions kept out of the design space.

- **Defeaturing for Design Progression**. If the Master-Model is built in a manner that reflects the design process, then traversing the stages in the process is just a matter of adjusting the Feature Tree. During preliminary design where the resultant geometry may be simple (or nonexistent) most of the branches of the tree are suppressed. As the design approaches the final intent, more and more of the details of the part are expressed by unsuppressing the branches. This will also require setting various parameters as their effects become active.

- **Defeaturing for different disciplines**. Suppressing branches of the Feature Tree can also be used to match the fidelity of the geometry to the analysis being performed. For example, if CFD is being used and the meshing scheme cannot handle fillets, then the fillets can be suppressed. This is a much simpler and more rigorous approach than trying to modify the fully expressed part after the fact (and it can be done automatically).

- **Parameter sweeps**. With the parameters defined in a meaningful manner, parameter studies can become as simple as setting a new value, having the CAD system regenerate the geometry and then analyzing the new instance. A complete design space can be mapped out from the complete set (or subset) of the parameters. This means that the process of automated design can be tracked and some insight gained into the design by visually tracing the selection of parameter values.

- **Feedback**. Traditional serial design settings, where one discipline performs its design and passes the results to the next, lacks the ability to easily recover from a conflict between the current discipline and the state of the geometry. This usually requires restarting the entire process. For example, aerodynamics designs a wing that does not contain enough space to support the structures. With an integrated continuous view of design and the appropriate parameterization, the shape can be modified at any time so that the aerodynamics team does not need to be explicitly involved in redesigning the wing.

In order for the proposed approach to be successful, the designer/CAD Operator must understand the nuances of the CAD package in use to robustly define Features that will persist across the family of parts. The parts must be put together with care, to ensure that the appropriate dimensions in the model are driven by meaningful parameters. Also, Features must be used in such as way as to allow later suppression and modifications to the CAD model as the design matures.

This also suggests a situation where a CAD model can be built up from a superset of the same geometric parameters as output by the class of codes which includes PASS. The execution of these mission-based conceptual design tools can be used to initially populate the parameter values and allow for a 3D realization of the concept ready for analysis. This also completely bypasses the traditional *artist rendering* and the CAD model can be used to visually communicate the design.

The CAD model should be constructed so that Master-Model effectively captures the decomposed intent of the design, starting with the most basic definition through to the finest manufacturing details. To achieve an even higher level of modularity and to more fully capture the design *intent*, complex parts should be modeled inside an assembly, especially for aerodynamically constrained applications. This is how the models seen is Figure 2 were constructed.

III. Conceptual-Design Geometry Tools

It is not uncommon to find customized tools that build up individual 3D components when sizing geometry in aircraft Conceptual Design. These tools are parametric and usually allow for rapid and interactive control of shapes (in comparison to a CAD regeneration). These Conceptual-Design Geometry Tools (CDGT), when used for building traditional air vehicles, typically have templates for components like fuselages, wings, tails, nacelles, etc., which can be scaled and positioned as needed to form the configuration.

It is difficult to replace Conceptual-Design Geometry Tools with CAD systems for a number of reasons. These include:

- **Cost of CAD licenses**. CAD systems are large scale complex software environments. Not only are the software licenses expensive, but the training costs for the learning of parametric CAD are often far greater. And in many cases, the application of CAD for some relatively simple conceptual tasks is akin to using a sledge hammer to drive in a brad.

- **Incompatible parameterization**. Not all parametric CAD systems are created equal, and the suite of features differ for each. It requires a great deal of specific CAD training to be able to compose the series of operations driven by the parameter set to form a particular design. In some instances it may not be possible to fully realize the desired parameterization due to a feature mismatch or a fundamental difference in construction or modification of geometry. A prime example of this would be *Free Form Deformation*.

- **Specialized construction**. The lofting procedures needed to generate a wing in a particular way may not exist in a particular CAD system. For example, the airfoils stacked to form a tapered wing can be either scaled and then interpolated, or alternatively interpolated and then scaled. The two procedures are not equivalent. And even with either approach, different interpolations can be used. Generally, splines are used in CAD systems for interpolation. However, there are other basis functions that could be used. For example, the use of Berstein Polynomials to parameterize airfoil shapes has been suggested by Kulfan.[12] Furthermore, the standard geometry file formats do not support these unusual geometry types. So to perform specialized constructions it would be necessary to convert (by additional fitting or approximating) to types such a BSplines or NURBS for transmittal to other parts of the MDAO process.

- **No interactivity of parametric changes**. Under some circumstances it is desirable to see updates of parameter changes in an interactive manner. The speed for any complex geometric build based on solid modeling cannot compete with simple surface manipulation.

Conceptual-Design Geometry Tools can be placed into two general classes based upon their underlying geometric control:

1. **Use of *Home Grown* geometry**. For simplicity and pointwise data accessibility, these tools are frequently limited to wire-frame or other simpler representations. Even if the geometry is represented by a series of BSpline patches that can be closed, usually no connectivity information is transmitted or even available. No component/component intersections are explicitly required. If multiple components are displayed simultaneously, the graphics engine performs the intersection and provides a display (but not the geometry) that appears correctly trimmed. Geometric output from these tools may use the IGES file format if the constructed geometry types are supported by the standard. Examples of this class include RAGE[13] and VSP.[14]

2. **Use of an existing geometry kernel**. This class of tools typically allows output from the kernel in the form of standard file formats, such as IGES or STEP. In cases where the underlying kernel is based on solid modeling, the resulting geometry is (curiously) not closed. This is an unnecessary limitation that makes the use of these tools in high-fidelity analysis not that much better than those discussed above. Examples are AML[15] and AMSketcher.[16]

Because the geometry from Conceptual-Design Geometry Tools is not appropriate for high-fidelity 3D analyses, a gap still exists between these tools and the analysis suites used in MDAO environments. With the historical legacy (and confidence) in these tools, and the difficulty or reluctance to move to CAD, the following question is raised:

How can we take the best of the CAD-based continuous solution and move the Conceptual-Design Geometry Tools towards high-fidelity Multidisciplinary Design, Analysis and Optimization?

IV. Bottom-Up and CSG Construction

Parametric CAD has a foundation based on Constructive Solid Geometry (CSG) concepts. In this way parts of arbitrary complexity can be generated as collections of simply abstracted operations ("features"), and at any step the resultant geometry is suitable for high-fidelity analysis. Furthermore, 3D meshes can be generated in an automated manner making this kind of geometry generation suitable for design settings. These favorable characteristics stem primarily from the use of solid models.

Incorporating CSG into the aircraft Conceptual-Design Geometry Tools so that solid models are produced is clearly a step that will allow CDGTs to be more easily connected with high-fidelity analysis. And once a Solid is produced, then the same types of CSG operations can be applied to the results of these tools to increase the level of geometric complexity that is a requirement of later stages of design. The process of producing a solid model from its constituent geometric entities is referred to as *Bottom-Up* construction. This can appear to be a tedious process where most of the effort is in the construction of Topology that uniquely defines the connectivity and hierarchy of the model. Only a few Topological entities actually hold geometric entities.

Clearly a system that provides the ability to perform Bottom-Up and CSG styles of geometric modeling would allow for CAD-like builds but also foster the ability to include customized construction. In the case of the Conceptual-Design Geometry Tools, having a *bridge* to solid modeling, will allow for these parametric tools to be a fully functional part of the larger high-fidelity MDAO environment. The concept is simple; build up Solid-based components as early as possible and then use CSG methods to assemble and integrate. The procedure envisioned involves some programming (either augmenting the Conceptual-Design Geometry Tools themselves or adding a filter that turns the output into a solid model). The obvious choice is an Application Programming Interface (API) that can be accessed by a number of programming languages (that are the same as those used by the CDGTs).

This is not unlike the geometry perspective used for the DaVinci Software[17] of CREATE-AV DoD Project. But what is suggested in this paper is a general geometry API that is generated specifically for the creation of Solids-based geometry, which remains closed during subsequent CSG operations. Unfortunately, the construction of a Solid Modeling geometry kernel is a daunting task, but clearly a prerequisite. This problem can be mitigated by the use of OpenCASCADE,[18] a fully functional Open-Source Solid modeling geometry kernel, that has the following characteristics:

- Support for manifold and non-manifold geometry.
- Has the ability to perform Bottom-Up construction.
- Has both CSG operations and other abstract *feature-like* construction methods.
- Can read and write IGES, STEP and native file formats.
- Is a fully Object-Oriented C++ API with about 17,000 methods in 2+ million lines of code!

The last point is an obvious problem. Both the C++ nature of OpenCASCADE and the level of programming complexity with the huge suite of methods makes the use of OpenCASCADE rather a difficult undertaking. Its lack of documentation adds to the enormous task of understanding this large, but capable software suite.

All of these issues provide the motivation for the design and the development of the software subsystem named **EGADS** (the **E**lectronic **G**eometry **A**ircraft **D**esign **S**ystem). This procedural-based API is:

- Object-Based built on top of OpenCASCADE.
- Has full support for current platforms (hardware & software). In particular, 32 and 64 bit versions are available for LINUX, MAC OSX and Windows machines. Other architectures are possible if OpenCASCADE can be ported.
- Open Source (LGPL v2.1) license so that it can obtain wide usage with few impediments.
- Has FORTRAN, C and C++ bindings.

A. EGADS Objects

It is a challenge to construct an object-based API that can be accessed by a number of procedural programming languages including FORTRAN. This is done in EGADS by providing the base object as a pointer to data that should not be decoded (dereferenced) except by the direct use of the API itself.

1. Blind Pointers

Internally the objects are made from bookkeeping-like information and the actual data that makes up the object. An **ego** (EGADS Object) is this *blind pointer* to the object data. For the FORTRAN bindings this is cast to a 64bit integer, which must be used as an INTEGER*8 variable. This allows for the C/C++ and FORTRAN API signatures to be almost identical. The data pointed to from an **ego** (minus the bookkeeping) consists of:

- Object type: One of 3 types of Geometry, 2 forms of Tessellation, Transform or the many kinds of Topology.
- Member type. Depends on the object type, for example for a Curve Object it can be LINE, CIRCLE, and etc.
- Object data. The specific information based on the object and member types.
- Optional Attributes.

2. Attributes on Objects

The ability to place useful information on any object within the system is critical for any system that requires associativity. Attribution allows for *ownership* to follow the object (or fragments of the object) through operations performed on the data objects. There can be any number of attributes attached to an object. Within EGADS an attribute consists of:

- Name. A unique character string identifying the attribute. There can only be a single attribute on an object with the name.
- Type. This refers to the kind of data assigned to the attribute. An attribute can only have one type: either Integer, Real or String.
- Length. This is the amount of data attached to the attribute. Character strings have an obvious length (the size of the string). Integer and Real attributes can have any (positive) number of entries.
- Value(s).

The scope of an attribute depends on the class of the owning object. Only those attributes on Topology will be persistent across EGADS sessions. And this will only take place if the Model data is written out in the native EGADS manner (i.e., not using IGES, STEP or OpenCASCADE file formats).

B. Geometry Objects

All BReps in EGADS are built from 0D (points in space), 2 types of 1D and 2D parameterized geometric entities. Points are not explicitly called out but become a part of the Topological entity called a Node. Curves and Surfaces are explicit Objects in EGADS which provides the services of *evaluation* and *inverse evaluation* (or "snaps") to the object in order to move between the physical and geometry's parameterized view.

- Curves.
  3D curve representations driven by a single running parameter (t). Supported member types include LINE, CIRCLE, ELLIPSE, PARABOLA, HYPERBOLA, OFFSET, BEZIER and NURBS (of which BSplines are a subset).

- PCurves (or Parameter Curves).
  Curves that sit on a specific surface also based on a single parameter (in this case $(u, v) = f(t)$). When used for an Edge the parameter range of t must match up with that of the 3D Curve and the (u, v) support defines the trace of the 3D Curve on the surface. The supported member types are the same as for the Curves.

- Surfaces.
  3D surface representations defined by 2 geometric parameters: $(x, y, z) = f(u, v)$. Supported member types include PLANAR, SPHERICAL, CONICAL, CYLINDRICAL, TOROIDAL, REVOLUTION, EXTRUSION, OFFSET, BEZIER and NURBS (which includes BSpline surfaces).

C. Topology Objects

When building the BRep up from primitives, the topological entities are created from the bottom of Table 1 up towards the top (one level at a time). When parsing existing topology (and attached geometry) the table is traversed from the appropriate level downward.

EGADS Topological Object	**OpenCASCADE term**	**Geometric Object**
Model	Compound Shape	
Body	Solid (or lesser Shape)	
Shell		
Face		Surface
Loop	Wire	* *see note*
Edge		Curve
Node	Vertex	[point]

Table 1. EGADS Topological Entities. *note: Loops may be geometry-free or have associated PCurves (one for each Edge) and the surface where the PCurves reside.

In general, topological entities (**ego**s) lower in Table 1 bound those entities directly above which produces an unambiguous hierarchy:

- Nodes.
  The simplest entity which is the topological equivalent to a point in 3 space.

- Edges.
  Most Edges have an underlying 3D curve which is bounded by 2 Nodes. The first Node is at t_{min} and the last is at t_{max}. An Edge can be the member type DEGENERATE which refers to a collapse of a surface's parameterization, ONENODE where the 2 Nodes refers to the same entity and the normal TWONODE type.

- Loops.
  Without a surface reference a Loop is an ordered collection of Edges with corresponding senses. In this case no Edges may be DEGENERATE.
  A Loop with a surface reference also contains PCurves, each associated with the Edge listed. The PCurve provides the mapping from the Edge's t to (u, v) on the surface. Here DEGENERATE Edges mark placeholders so that the PCurve can properly bound the surface's parametric mapping. It should be noted that a Edge can be found in a Loop twice (once in the positive orientation and the other negative). This occurs for closed surfaces such as cylinders. For this situation there are two different PCurves, each representing the different limits of the periodic nature of the surface.
  The Loop may be of the OPEN or CLOSED (when it ends at the Node where it started) member type.

- Faces.
  Faces refer to a surface which is bounded by one or more CLOSED Loops. There must be one *outer* Loop and all the others are *inner* Loops (or holes). The reference geometry of the Loops must match the Face's surface. Attributes on Faces are robustly tracked through all high-level (*Feature*-based) construction. For example when using the Solid Boolean Operators, the resultant BRep(s) have the Face attributes from the source bodies. If all Faces were marked in the input BReps, then all Faces in the output will also be marked.

- Shells.
  Shells are simply a collection of orientated Faces. This represents the CLOSED type if all Edges found in the Face's Loops are accounted for twice (unless DEGENERATE).

- Bodies.
  This represents a functional aggregation of entities that can be used as the group. This is usually the terminal condition for Bottom-Up construction. There are four member types:

  1. WireBody – A single Loop which can be OPEN or CLOSED. These can be used as input to Extrude, Revolve and Lofting operations where the result is a SheetBody.
  2. FaceBody – A single Face (which can be viewed as a subset of an OPEN SheetBody). These can be used as input to Extrude, Revolve and Lofting operations where the result is a SolidBody.
  3. SheetBody – A single Shell that can be OPEN or CLOSED.
  4. SolidBody – One or more CLOSED Shells. There is always a single *outer* Shell with any number of *inner* Shells that represent the removal of material from the interior of the Solid.

  All but the SolidBody refer to non-manifold BReps.

- Models.
  This is the top-level EGADS container which can hold any number of Bodies. This is what the system reads and writes to disk (which includes the import and export of IGES and STEP file formats). In order to deal with fully connected non-manifold entities in the "Body" context describe above, it is appropriate to have multiple Bodies within a single Model that share Topological objects. For example a SolidBody that reflects the OML of a aircraft could share the wing trailing-edge Edges with a SheetBody that represents the wake surface. There is an EGADS function that allows for the testing of *equivalency* of objects so that the wake surface could, in a sense, be reconnected to the manifold representation of the aircraft.

D. EGADS Tessellation & Tessellation Objects

The CAPRI approach that provides a rich and associative tessellation of a BRep coupled to the geometry is mimicked in EGADS. When generating a discrete representation of a Body, this allows for a view of trimmed geometry that is topologically closed though may be open at machine precision. A Tessellation object can be generated for any Body with the same adjustable parameters as used in CAPRI. The resultant discretization can be examined a Face at a time. The triangulation of trimmed surfaces includes a single discretization of the Edges (for both Faces that are trimmed). Therefore, for SolidBodies when the Face triangulations are put together they form a completely closed manifold triangulation.

To support analyses that may need a quadrilateral discretization of the Body (for example: panel codes), EGADS provides non-automatic techniques to place patches on the BRep. A template scheme is used (like the method in CAPRI for Anisotropic Triangulating[9]) that patches a single Looped Face where 3 or 4 sides can be identified. The point counts on opposing sides are used (from the Body tessellation) to select the template and then filled with as many as 17 larger unstructured quadrilateral patches. These are then subdivided in a regular manner based on the point counts of the exposed sides. This is not automatic because there are situations that can not be templated due to an odd number of points on the larger Loop.

EGADS provides functions that allow for the movement, addition and removal of Edge discretization points so that the quad-patching can provide surface meshing commensurate with the task at hand.

Unlike CAPRI (which does not support Bottom-Up construction) there is the ability to generate tessellations for Geometry alone. This untrimmed discretization is useful for viewing the data in EGADS so that orientation and extent can be examined in order to support BRep building.

E. Sample API Functions

There are approximately 60 functions currently in EGADS (which provides a marked contrast to the number of OpenCASCADE methods). The following is obviously not the complete set. These C-like prototypes are intended to give the reader a sense of the API.

1. Open

Creates and returns an EGADS Context object. This special object contains the *context* in which subsequent EGADS functions can operate. There can be multiple contexts in a single session.

```
icode = EG_open(ego *context)
```

2. Load Model

Loads and returns a Model object from disk and puts it in the specified EGADS context.

```
icode = EG_loadModel(ego context, int flags, char *name, ego *model)

flags: 0 - Split periodics
       1 - Dont split closed and periodic entities
name:  Load by extension:
           igs/iges
           stp/step
           brep  (for native OpenCASCADE files)
           egads (for native files with persistent Attributes, split ignored)
```

3. Get Information on a Topological Object

```
icode = EG_getTopology(ego object, ego *ref, int *oclass, int *mtype, double *data,
                       int *nchild, ego **pchldrn, int **psenses)

ref:       is the reference geometry object (if none this is returned as NULL)
oclass:    is Node, Edge, Loop, Face, Shell, Body or Model
mtype:     for Edge is TWONODE, ONENODE or DEGENERATE
           for Loop is OPEN or CLOSED
           for Face is either SFORWARD or SREVERSE
           for Shell is OPEN or CLOSED
           Body is either WireBody, FaceBody, SheetBody or SolidBody
data:      will retrieve at most 4 doubles:
           for Node this contains the [x,y,z] location
           Edge is the t-start and t-end (the parametric bounds)
           Face returns the [u,v] box (the limits first for u then for v)
nchild:    number of children (lesser) topological objects
pchldrn:   is a returned pointer to the block of children objects.
psenses:   is the returned pointer to a block of integer senses for the children.
```

4. Create a Topological Object

```
icode = EG_makeTopology(ego context, ego ref, int oclass, int mtype, double *data,
                        int nchild, ego *chldrn, int *senses, ego *object)

ref:       is the reference geometry object (if none this should be NULL)
oclass:    is Node, Edge, Loop, Face, Shell, Body or Model
mtype:     for Edge is TWONODE, ONENODE or DEGENERATE
           for Loop is OPEN or CLOSED
           for Face is either SFORWARD or SREVERSE
           for Shell is OPEN or CLOSED
           Body is either WireBody, FaceBody, SheetBody or SolidBody
data:      may be NULL except for:
           Node this must contain the [x,y,z] location
           Edge is the t-start and t-end (the parametric bounds)
nchild:    number of children (lesser) topological objects
chldrn:    the block of children objects.
senses:    the block of integer senses for the children.
```

5. Create a Geometric Object

```
icode = EG_makeGeometry(ego context, int oclass, int mtype, ego rGeom,
                        int *pinfo, double *prv, ego *geom)

context:  the Context object used to place the result
oclass:   PCurve, Curve or Surface
mtype:    PCurve/Curve
                LINE, CIRCLE, ELLIPSE, PARABOLA, HYPERBOLA, TRIMMED,
                BEZIER, BSPLINE, OFFSET
          Surface
                PLANE, SPHERICAL, CYLINDER, REVOLUTION, TORIODAL,
                TRIMMED, BEZIER, BSPLINE, OFFSET, CONICAL, EXTRUSION
rGeom:    is the reference geometry object (if none use NULL)
pinfo:    is a pointer to the block of integer information. Required for
          either BEZIER or BSPLINE.
prv:      is the pointer to a block of double precision reals. The
          content and length depends  on the oclass/mtype.
geom:     is the resultant new geometry object
```

6. Fit or Approximate Geometry

Computes and returns the resultant object created by approximating a BSpline. If the tolerance is zero for a surface then the data is fit.

```
icode = EG_approximate(ego context, int mDeg, double tol, int *sizes, double *xyz,
                       ego *geo)

context:  the Context object used to place the result
mDeg:     is the maximum degree used for the approximation [3-8]
          Note: fits are always cubic.
tol:      is the tolerance to use for the BSpline approximation procedure
sizes:    a vector of 2 integers that specifies the size and dimensionality of
          the data. If the second is zero, then a curve is assumed and the first
```

```
            integer is the length of the number of [x,y,z] triads. If the second
            integer is nonzero then the input data reflects a 2D map of coordinates.
  xyz:      the data to fit (3 times the number of points in length)
  geo:      the returned approximated (or fit) BSpline resultant object
```

7. Make a Face

Creates a simple Face from a Loop or a surface. This also creates any required Node(s), Edge(s) and Loop(s).

```
  icode = EG_makeFace(ego object, int mtype, double *data, ego *face)

  object:  either a Loop (for a planar cap) or a surface with [u,v] bounds
  mtype:   is the desired orientation of the Face
  data:    may be NULL for Loops but must be the limits for an input surface
  face:    the resultant returned topological FACE object
```

8. Solid Boolean Operator

Performs the Solid Boolean Operations (SBOs) on the source Body Object (that has the type SolidBody). The tool object types depend on the operation.

```
  icode = EG_solidBoolean(ego src, ego tool, int oper, ego *model)

  src:        the source SolidBody object
  tool:       the tool object: either a SolidBody for all operators  -or-
              a Face/FaceBody for Subtraction
  oper:       Subtraction, Intersection or Union
  model:      the resultant Model object (this is because there may be multiple
              bodies from either the subtraction or intersection operation).
```

9. Save Model

Saves an EGADS Model object to disk.

```
  icode = EG_saveModel(ego model, char *name)

  name:  Save by extension:
             stp/step
             brep  (for native OpenCASCADE files)
             egads (for native files with persistent Attributes)
```

10. Close

Closes and destroys all data associated with an EGADS Context.

```
  icode = EG_close(ego context)
```

V. Example Build using EGADS

After a traditional conceptual design phase, a typical output is a basic 3-view and a few cross-sections, together with tables of key physical parameters. An example is shown in Figure 3.

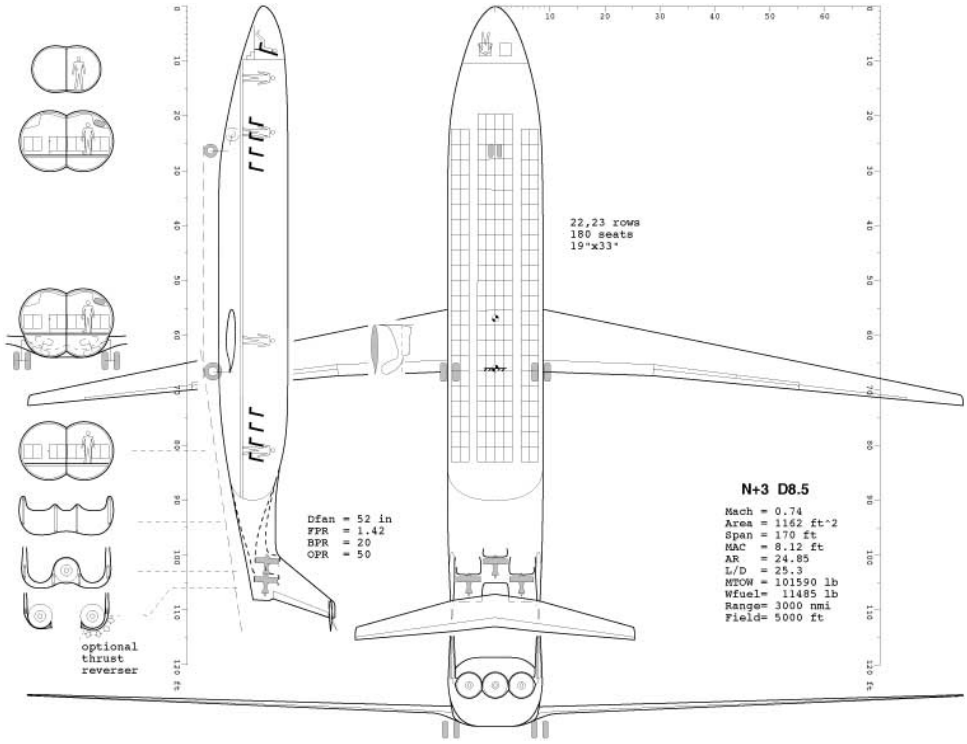


Figure 3. Three-view of MIT's D8.5 ("double bubble") series concept for NASA's N+3 program.[19] The aircraft is a Boeing 737-800 replacement, with dramatic fuel-burn reductions from new technology and especially from a new fuselage configuration concept.

In order to communicate the concept to others the next step in the traditional design setting is to pass the concept on to a *artist* to have a rendering of the aircraft produced. An example is depicted in Figure 4.

Figure 4. Artist's Rendering of MIT's D8.5 concept aircraft.

For performing panel method calculations on the D8 aircraft, a basic wire-frame definition was generated by using a home-grown Conceptual-Design Geometry tool based on stacking airfoils and fuselage cross-sections, and using suitable linear or bi-cubic interpolation to generate the x, y, z wire-frame point. Figure 5 displays each wire-frame component in a different color. It should be noted that the aft engine nacelles have

been removed to simplify the construction of this wire-frame model.

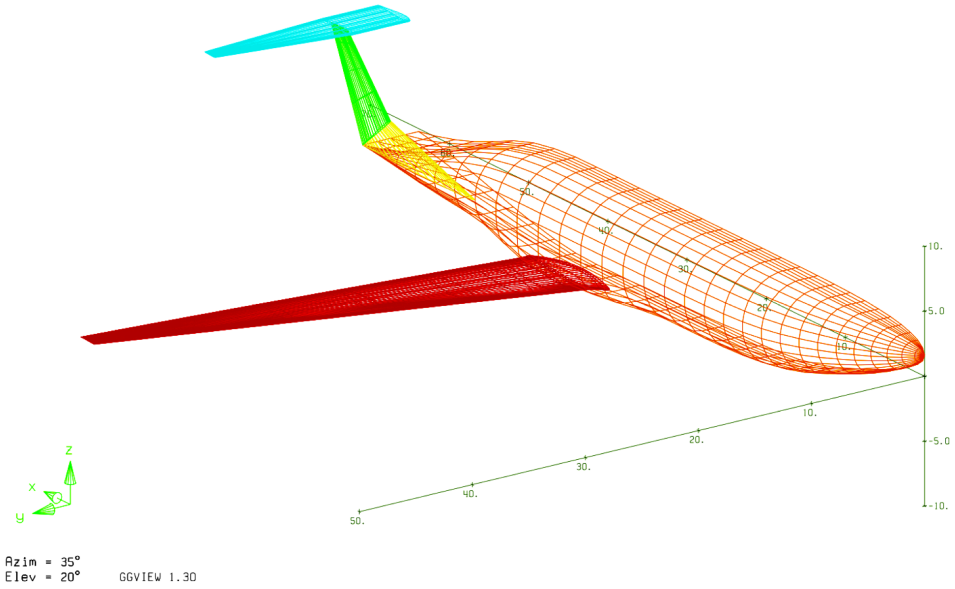


Figure 5. Wire-frame of 1/2 of the D8 model used for panel calculations.

Low-order panel methods in particular are extremely insensitive to geometry quality, and most can easily tolerate defects such as poor resolution, ill-defined curvatures, and even lack of tip endcaps, with little effect on the overall forces and moments. Hence, the simple wire-frame geometry is quite adequate for initial panel calculations to do basic checkout of the design. In particular, wing spanload distributions and induced drag, shear and bending moment diagrams, and also the stability and control characteristics can all be estimated at this stage.

Figure 6. Wind tunnel model of the D8 concept aircraft. Note that the engines have not been realized in this geometry.

A typical next step in development of the OML and initial structure sizing is to use high-fidelity Navier-Stokes CFD methods, commercial FEA methods, or wind tunnel models constructed via CNC or other numerical fabrication methods. A wind tunnel model of the D8 concept is shown in Figure 6. For these design

phases the wire-frame definition and the artist's rendering model are generally inadequate. A watertight geometry model is required instead, and here is one instance where EGADS can come into play.

The same wire-frame input (as seen in Figure 5) was used to directly generate a series of solid components using a small number of EGADS calls. The `EG_approximate` function was used to make smooth BSpline surfaces from the wire-frames for all of the components. Faces were directly constructed from these surfaces by the `EG_makeFace` function and the components were closed by Bottom-Up construction calls (`EG_makeFace`, `EG_makeGeometry` and `EG_makeTopology`). All were fused together by applying the union Solid Boolean Operator (`EG_solidBoolean`) producing a single solid appropriate to 3D high fidelity analysis. The SolidBody representing 1/2 of the D8 can be seen in Figure 7.

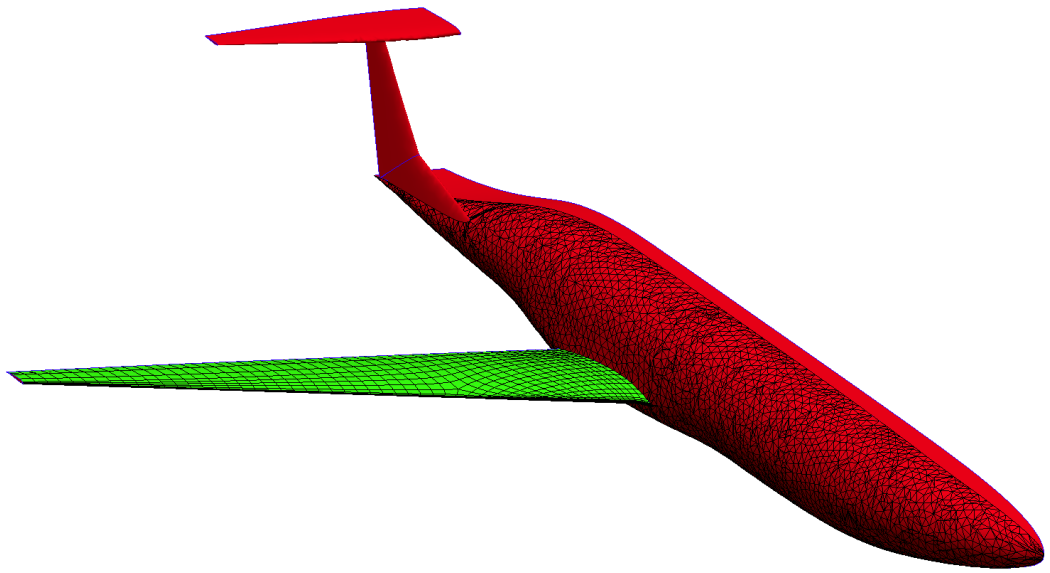


Figure 7. EGADS solid model produced from the data provided from the wire-frame components shown in Figure 5.

Figure 7 displays a triangulation of the middle panel of the fuselage and quadrilaterals for the wing (in green). The tessellation is watertight even with this mixed discrete representation because the Edge discretization is consistently used for either the triangulation and/or the quadrilaterals. This solid model can now be imported into many diffferent Navier-Stokes solvers, FEA packages, CAM systems, or applications using STL files such as 3D printers.

VI. Conclusions

The construction of geometry is usually the most overlooked part of any MDAO setting (whether for aircraft design or other manufacturing tasks). This is an odd situation, because in most cases the ultimate goal is a best *shape* that fits the mission. This paper discusses two modes of model construction for design. Each has strengths and weaknesses.

A. CAD-based Model Construction

Using a CAD system requires the purchase of large-scaled licensed software. These systems have quite a steep learning curve in order for an operator to become effective at the use of their capabilities. Though all current systems are parametric and can use CSG constructs, they all have the ability to generate geometry in a

legacy mode akin to drafting. Most CAD operators one finds in larger aerospace manufacturing environments have not transitioned from drafting to parametric CAD. So it must be noted that the use of these CAD models (from this drafting mode of construction) is wholly inconsistent with automated design. There are no parameters that could define a design space and the rebuild of geometry is therefore not automatically possible.

The use of parametric CAD requires a very different mindset that that of drafting. It is analogous to object-oriented programming and requires thinking about the construction as the series of CSG operations or "features". The *parameters* are defined by parts of the procedure that dimension or size the extent of the operation. In general, a proficient parametric CAD operator can put a part together in much less time than the individual performing drafting. This is because the details of the sizing and shape of the component can be deferred, all that is required is that the part be flexible and malleable and can regenerate under the range of parameters of interest.

It is best if the CAD designer/operator knows how the part (or assembly) will be used and by which analysis suites. In this way the geometry can be made appropriate for each discipline. Preparing the geometry for pre-processing can be easily accomplished by featuring/defeaturing the build tree and the use of the Solid Boolean Operators (for example, to define the fluid domain if given the OML). In practice this is rarely done. Generating geometric models that can be used in this manner is currently viewed as an *art*. There is little training for using Parametric CAD in MDAO and no *best practices guides*.

Using CAD from as early as the Conceptual Design phase has, as its most important advantage, the ability to construct a seamless design process that can be used through manufacturing. It provides a technique where the design intent can be found in a single place (the CAD parts). No translations of importation of geometry is necessary (which would potentially loose the parametric nature of the component). PLM or PDM systems can be used to track design and manufacturing decisions throughout the lifetime of the design.

B. EGADS Model Construction

EGADS provides a solid modeling geometry kernel that supports both Bottom-Up construction as well as the ability to perform Constructive Solid Geometry operations. This is the proper foundation for the construction of *closed* 3D geometry (a prerequisite for high fidelity analysis). The API has been designed for integration within the suite of tools that are used specifically for generating geometry for conceptual aircraft design (but may be more generally useful). The integration can be fairly simple by taking the parametrically built components (i.e., wings, fuselages, tails, nacelles, and etc.), using the Bottom-Up functions to construct the topology to *close* the model. At this point the component could be written out (in a file format that supports solids) or used by the higher-level CSG operations (such as the Solid Boolean Operator – Union, filet, and etc.) to continue the construction.

Because EGADS supports writing geometry out in the STEP file format, the model(s) can be used directly by CAD systems via the import of this file type. In this case there would be no loss of geometric data or fidelity but the design intent is lost. Here the *parameters* and the construction recipe exist only in the Conceptual-Design Geometry Tool and would need to be re-executed under a design change. Clearly, taking this direction generates a process that is more fragmented (in terms of geometry usage) and therefore more difficult to manage. It should also be noted that if parametric sensitivities are required by the optimization portion of the MDAO process, then having the build defined in multiple places makes the calculation of this matrix much more difficult than it is when the design intent is self-contained.

Going this route, one needs to always be cognizant of how much customized software is being generated. There is an extraordinary amount of work required to build up the software infrastructure able to construct the kind of assembly-based parametric model (as seen in Figure 2(b)) so that aircraft structural layout properly *morphs* during parametric changes. It never makes sense to expend the kind of resources it would take to basically write a CAD system.

The work in EGADS continues as it becomes one of a number of ways to manage geometry within the OpenMDAO framework.[20] This allows for a open-source geometry solution that embeds geometry at the center of the MDAO process.

Acknowledgements

The authors would like to thank Harold Youngren of Aerocraft, Inc. for helpful discussions and input into the formulation of EGADS.

References

[1]Kroo, I. M., "An Interactive System for Aircraft Design and Optimization," No. AIAA 92-1190, Aerospace Design Conference, American Institute of Aeronautics and Astronautics, Irvine, California, 3-6 February 1992.

[2]Jouannet, C. and Krus, P., "Direct Simulation Based Optimization for Aircraft Conceptual Design," No. AIAA 2007-7827, 7th AIAA Aviation Technology, Integration and Operations Conference (ATIO), American Institute of Aeronautics and Astronautics, Belfast, Northern Ireland, 18-20 September 2007.

[3]Lazzara, D. S., Haimes, R., and Willcox, K., "Multifidelity Geometry and Analysis in Aircraft Conceptual Design," 19th AIAA Computational Fluid Dynamics Conference, American Institute of Aeronautics and Astronautics, San Antonio, TX, 22-25 June 2009.

[4]Amadori, K. and Jouannet, C., "A Framework for Aerodynamic and Structural Optimization in Conceptual Design," No. AIAA 2007-4061, 25th AIAA Applied Aerodynamics Conference, American Institute of Aeronautics and Astronautics, Miami, Florida, 25-28 June 2007.

[5]Lazzara, D. S., Parham, J. B., and Haimes, R., "On Structural Layout using Multifidelity Geometry in Aircraft Conceptual Design," No. AIAA 2010-1316, 46th AIAA Aerospace Sciences Meeting, American Institute of Aeronautics and Astronautics, Orlando, FL, 4-7 January 2010.

[6]Haimes, R. and Follen, G., "Computational Analysis PRogramming Interface," Proceedings of the 6th International Conference on Numerical Grid Generation in Computational Field Simulations, July 1998.

[7]Aftosmis, M. J., Delanaye, M., and Haimes, R., "Automatic Generation of CFD-Ready Surface Triangulations from CAD Geometry," No. AIAA 99-0776, 37th AIAA Aerospace Sciences Meeting and Exhibit, American Institute of Aeronautics and Astronautics, Reno, Nevada, 11-14 January 1999.

[8]Haimes, R. and Aftosmis, M. J., "On Generating High Quality "Water-tight" Triangulations Directly from CAD," Proceedings of the 8th International Conference on Numerical Grid Generation in Computational Field Simulations, 2002.

[9]Haimes, R. and Aftosmis, M. J., "Watertight Anisotropic Surface Meshing Using Quadrilateral Patches," Proceedings of the 13th International Meshing Roundtable, 2004.

[10]Jones, W. T., Personal Communication, 2010.

[11]Jones, W. T., Lazzara, D. S., and Haimes, R., "Evolution of Geometric Sensitivity Derivatives from Computer Aided Design Models," No. AIAA 2010-9128, 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference, American Institute of Aeronautics and Astronautics, Fort Worth, TX, 13-15 September 2010.

[12]Kulfan, B. M., "Universal Parametric Geometry Representation Model," *Journal of Aircraft*, Vol. 45, No. 1, January-February 2008, pp. 142–158.

[13]Rodriguez, D. L. and Sturdza, P., "A Rapid Geometry Engine for Preliminary Aircraft Design," No. AIAA 2006-0929, 44th AIAA Aerospace Sciences Meeting and Exhibit, American Institute of Aeronautics and Astronautics, Reno, Nevada, 9-12 January 2006.

[14]Fredericks, W. J., Antcliff, K. R., Costa, G., Deshpande, N., Moore, M. D., San Miguel, E. A., and Snyder, A. N., "Aircraft Conceptual Design Using Vehicle Sketch Pad," No. AIAA 2010-0658, 46th AIAA Aerospace Sciences Meeting, American Institute of Aeronautics and Astronautics, Orlando, FL, 4-7 January 2010.

[15]TechnoSoft, "Adaptive Modeling Language," http://www.technosoft.com/aml.php.

[16]TechnoSoft, "AMSketcher," http://www.technosoft.com/amsketcher.php.

[17]Roth, G. L., Livingston, J. W., Dailey, C. J., and Cline, A. B., "Addressing Geometry Needs of Systems Engineering with DaVinci Software," No. AIAA 2011-1107, 47th AIAA Aerospace Sciences Meeting, American Institute of Aeronautics and Astronautics, Orlando, FL, 4-7 January 2011.

[18]Open CASCADE S.A.S., "OpenCASCADE," http://www.opencascade.org.

[19]Drela, M., "N+3 Aircraft Concept Designs and Trade Studies – Appendix," Tech. Rep. NASA CR-2010-216794/VOL2, NASA, 2010.

[20]Gray, J., Moore, K. T., and Naylor, B. A., "OpenMDAO: An Open Source Framework for Multidisciplinary Analysis and Optimization," No. AIAA 2010-9101, Fort Worth, Texas, 13-15 September 2010.