Computational Aircraft Prototype Syntheses (CAPS) Overview

December 6, 2022

0.1 Main Page

0.1.1 Introduction to CAPS

Aerospace vehicle design can be described as an evolutionary process of gathering information to make informed decisions. Meticulous application of this process involves numerous simulations covering many disciplines and fidelity levels. A design team needs to be able to easily increase or decrease fidelity as they gather more information about a particular design. To this end a geometry system that can support multi-disciplinary, multi-fidelity analysis from a single source is required. The Computational Aircraft Prototype Syntheses (CAPS), which is a part of the Engineering Sketch Pad (ESP) [12], satisfies the above by combining proven computational geometry, meshing, and analyses model generation techniques into a complete browser-based, client-server environment that is accessible to the entire design team of an aerospace vehicle. CAPS links analysis and meshing disciplines to any ESP geometry model via dynamically-loadable Analysis Interface Module (AIM) plugins. CAPS is accessed from either a browser-based user interface and or Multi-Disciplinary Analysis and Optimization (MDAO) framework through a programming interface. Here, we describe the fundamental building blocks of CAPS in ESP. The paradigm shift of creating geometry for multi-fidelity design is described in detail and represented in ESP scripts. We then demonstrate the use of this multi-fidelity geometry to support multi-fidelity, multi-physics analysis including discipline coupling.

The conceptualization and philosophy related to CAPS program is described in [3]. This document introduces main concepts and requirements to use CAPS. Before beginning the user should be familiar with ESP and be comfortable with creating parametric geometry inputs in a $*$.csm file format. For the user not familiar with ESP at a command prompt execute the following.

```
» serveESP
```

Click on the **Help** button on the upper left of the screen. This will bring up the **Engineering Sketch Pad (ESP) Version 1.22** documentation, that includes manual and tutorials. After completing the tutorials return to this CAPS Overview!

0.1.2 ESP Enhancement: Integrated and Collaborative Design Environment

This release includes an innovative way to develop ESP and CAPS scripts. A few of the highlights are: The scripts developed using this IDE are version controlled, both analysis and geometry scripts can be accessed and modified in the same familiar environment, and can be accessed in a multi-user setup for collaboration [6]. Please refer to Tutorial 6 in the ESP manual, which is a walk-through for this new capability.

0.1.3 Executive Overview

The primary programmatic access point into CAPS is through the **CAPS Executive.** It is envisioned that there will be 3 different approaches to using CAPS. One way is to interactively build a model and exercise the build to examine aspects such as the design sensitivities. Another scenario is to run one or more analysis packages interactively. Both of these approaches use an enhanced version of ESP (within a Web Browser) to interact with CAPS directly. The last approach has CAPS driven by an optimizer or by an MDO framework, such as MSTC Engr [16], Model↩ Center [23] or OpenMDAO [14]. The access, under all of these cases, is through the CAPS API, which in essence, is the portal to all of the CAPS functionality (the lower 2 dotted arrows as seen at the left of the CAPS Block Diagram Figure.

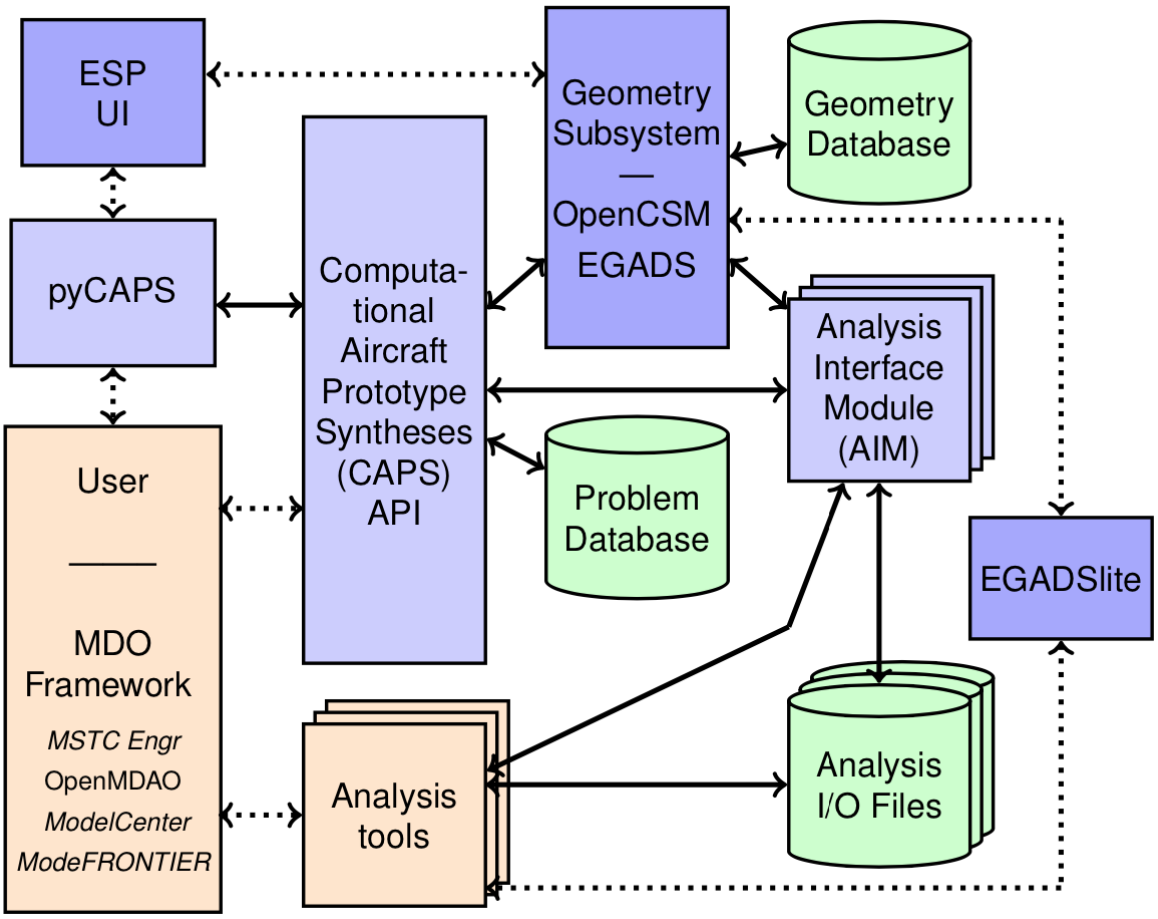

Figure 1 CAPS Block Diagram

The CAPS API is 'object-based' very much in the same manner as both EGADS [13] and OpenCSM [5]. 'Object-based' (unlike object-oriented) is a technique that allows for the use of objects in traditional procedural-based programming. This is done with opaque (or blind) pointers that the API functions parse and then perform the desired functions driven by the type of object input. The hierarchy of the CAPS objects can be seen in CAPS Objects Table and include the following types:

capsProblem. The Problem is the top-level container for a single mission. It maintains a single set of interrelated geometric models, analyses to be executed, connectivity and data associated with the run(s), which can be both multi-fidelity and multi-disciplinary. Various data entries can be connected via linkage ports found in all input objects. There can be multiple Problems in a single execution of CAPS and each Problem is designed to be thread safe allowing for multi-threading of CAPS at the highest level.

capsValue. A Value Object is the fundamental data container that is used within CAPS. It can represent inputs to the Analysis and Geometry subsystems and outputs from both. Also, Value Objects can refer to mission parameters that are stored at the top-level of the CAPS database. The values contained in any input Value Object can be bypassed by the linkage connection to another Value (or DataSet) Object of the same shape. Attributes are also cast to temporary (User) Value Objects.

capsAnalysis. The Analysis Object refers to an instance of running an analysis code. It holds the input and output Value Objects for the instance and a directory path in which to execute the code (though no explicit execution is initiated). Multiple various analyses can be utilized and multiple instances of the same analysis can be handled under the same Problem.

capsBound. A Bound is a logical grouping of BRep Objects that all represent the same entity in an engineering sense (such as the *upper surface of the wing*). A Bound may include BRep entities from multiple Bodies; this enables the passing of information from one Body (for example, the *aero OML*) to another (the *structures IML*).

capsVertexSet. A VertexSet is a connected or unconnected group of locations at which discrete information is defined. Each connected VertexSet is associated with one Bound and a single Analysis. A VertexSet can contain more than one DataSet. A connected VertexSet can refer to 2 differing sets of locations. This occurs when the solver stores its data at different locations than the vertices that define the discrete geometry (i.e. cell centered or non-isoparametric FEM discretizations). In these cases, the solution data is provided in a different manner than the geometry itself.

capsDataSet. A DataSet is a set of engineering data associated with a VertexSet. The rank of a DataSet is the (user/pre)-defined number of dependent values associated with each vertex; for example, scalar data (such as pressure) will have rank of one and vector data (such as displacement) will have a rank of three. Values in the DataSet can either be deposited there by an application or can be computed (via evaluations, data transfers, or sensitivity calculations).

Object	SubTypes	Parent Object
capsProblem	Parametric, Static	
capsValue	GeometryIn, GeometryOut, Branch, Parameter, User	capsProblem, capsValue
capsAnalysis		capsProblem
capsValue	AnalysisIn, AnalysisOut	capsAnalysis, capsValue
capsBound		capsProblem
capsVertexSet	Connected, Unconnected	capsBound
capsDataSet	User, Analysis, Interpolate, Conserve, Builtin, Sensitivity	capsBound

Figure 2 CAPS Objects

0.1.4 Analysis Subsystem

The Analysis Subsystem is where analysis data is prepared, held, queried, and where meshing is either accomplished or inputs are prepared for the use of a stand-alone grid generation package. A BRep at the proper level of fidelity (optionally with its tessellation) can be retrieved from the Geometry Subsystem, which is used for the meshing process. Again, to maintain the flexibility of CAPS, all specific analysis packages have (at least) one associated plug-in to perform the associated analysis. Multiple plug-ins may be attached to a single analysis package if the analysis has different execution modes (with different input requirements) and/or requiring differing levels of geometric fidelity. See the discussion of the AIM plug-ins below.

This subsystem is responsible for parsing the analysis output and aggregating the results to be passed to the CAPS Executive or maintained within this subsystem to pass on to another analysis module (in a multi-disciplinary setting). The plugin also handles retrieving the salient outputs (that can be constructed as objective functions) from the analysis.

0.1.4.1 Analysis Interface Module (AIM) Plug-ins

CAPS connects applications to ESP through Analysis Interface Modules or AIMs. These applications to date have involved meshing or specific analysis tools. Each AIM supplied has a specific tutorial that enables a user to use its intended applications. Additional information is provided in **AIM Overview** regarding the currently supported analysis tools.

For more information on each of the AIMs please see the specific tutorial for the AIM you are interested in. This CAPS overview will continue to introduce the overall features of CAPS. If you are new to CAPS it is recommended that you finish this overview before jumping to the specific AIM tutorials.

The following functions are a part of any AIM plug-in:

Attribute/Input Checking. This AIM function is invoked before any mesh/input file generation to ensure that all of the required data can be found.

Meshing. The input BRep and/or tessellation are used to either perform the meshing directly (if possible or the mesh system has an API) or to provide input to a grid generator. Note that the mesh vertices that sit on geometry (as described in the input BRep) need to be associated back to the geometry. This is important for generating parametric sensitivities and performing conservative data fitting. (*Most stand-alone grid generation systems maintain this data internally but do not make it available as output. Any attempt to re-associate this data by inverse evaluations is slow and not robust.*)

Analysis Input File(s) Generation. The input values and attributes found on the geometry are used to construct and generate the input file(s) required to run the analysis.

Output file parsing. This is required to get performance data, displacements, pressures or other information required to be used as input to another analysis module or to inform the optimizer of the objective functional value(s).

Conservative Data Transfer Functions. [7] In order to perform the interdisciplinary coupling in a conservative manner, functions that compute interpolation within a surface element, integration of quantities over an element (and their backward or dual variants) are needed.

0.1.5 pyCAPS

As previous detailed, CAPS is designed to be incorporated into a larger MDO infrastructure; as such pyCAPS [11] provides a light weight, Python-based framework that logically ties together features of the CAPS 'C' API to enable rapid generation of problems in the Python environment. To date all essential features of the CAPS API have been utilized in pyCAPS. Additional features can be easily ported and will be made available upon request.

In its current state, pyCAPS consists of a single parent class, 'capsProblem', and three children classes: 'caps↩ Analysis', 'capsGeometry', and 'capsBound'. The capsProblem class controls the overall workflow for a given problem and contains functions to load the geometry, load AIMs, setup data transfers, etc. The children classes provide additional functionality to the capsProblem class for dealing with specific aspects of their respective domains. Multiple analysis (capsAnalysis class) and data transfer (capsBound class) objects may be loaded into a single capsProblem object and are book-kept by the class in a dictionary format for easy reference. The following outlines additional details for the children classes:

capsAnalysis A single instance of the capsAnalysis class is synonymous with an instance of a loaded AIM. This class provides the user the ability to set input values (e.g., Mach number, angle of attack), retrieve output variables (e.g., lift or drag coefficient), and other functionality for a given analysis plug-in.

capsGeometry This class consists of functions to interact with the geometry model (e.g., set a geometric design parameter) loaded into a given capsProblem.

capsBound A single instance of the capsBound class corresponds to an instance of a capsBound, with caps↩ DataSets being referenced in a dictionary format. The primary function of the capsBound class is to execute the CAPS routines needed to initiate a data transfer to take place for given variable (i.e., capsDataSet).

Please refer to pyCAPS documentation for a more in depth discussion on its use.

0.1.6 Geometry Attributes

In CAPS, there are two sources of data to perform a computational analysis. These are the geometry itself, and AIM input values. For example, think of a surface on a wing that is a boundary condition for a CFD analysis. This surface can be identified with an attribute that gives it a name. For example:

```
ATTRIBUTE capsGroup $wingSurface
```

∗*Recall that in ESP a '$' designates a string*

When this attribute is attached to a specific piece of geometry inside the ∗.csm input then it can have more information associated with it inside an AIM. For example, if we use the CFD solver FUN3D as an example a pyCAPS entry that defines the above capsGroup attribute may be:

```
bc1 = {"bcType" : "Viscous", "wallTemperature" : 1}
bc2 = {"bcType" : "Inviscid", "wallTemperature" : 1.2}
fun3d.input.Boundary_Condition = {"WingSurface": bc1,
                                  "WingTip"    : bc2,
                                  "Farfield"   : "farfield"}
```

In the entry above two other capsGroup entries are defined *WingTip* and *Farfield*. This information will now naturally propagate to any mesh and FUN3D input. This is the point of attribution. It allows information to be located (on the geometry) and defined in a single place.

Every AIM uses attributes and AIM inputs to enable it. These are given in detail in each individual AIM's documentation. AIM documentation includes all the attributes, AIM Inputs, AIM Outputs and Example geometry and pyCAPS inputs to use them.

Many attributes are common across multiple AIMS. Below is a brief description of attributes that are commonly used in many AIMs:

capsGroup. The capsGroup attribute is used to logically group components together and or give a string name to a specific portion of geometry. This is commonly used to identify areas where specific boundary conditions will be applied. The string name used is entirely up to the user. This attribute is used by almost every AIM.

capsMeshLength. The capsMeshLength attribute is used to scale meshing input paramters that are in unites of length. For example, if a mesh generator has a maximum spacing input parameter and capsMeshLength is the mean aerodynamic chord, a maximum spacing of 0.05 will results in elements no longer than 5% of the mean aerodynamic chord.

capsType. The capsType attribute is used to separate lifting surfaces and body surfaces in low fidelity aerodynamic tools. This attribute must follow the naming convention defined for a specific AIM of interest. Example AIMs include Friction and AWAVE.

capsLength. Some analysis tools require that input files be created in a specific unit system. To accomplish this the AIM requires the unit system that the ∗.csm geometry is created in. This attribute defines this unit system. All conversion is handled by the AIM internally.

capsAIM. The CSM script generates Bodies which are designed to be used by specific AIMs. The AIMs that the Body is designed for is communicated to the CAPS framework via the "capsAIM" string attribute. This is a semicolon-separated string with the list of AIM names. Thus, the CSM author can give a clear indication to which AIMs should use the Body. For example, a body designed for a CFD calculation could have:

```
ATTRIBUTE capsAIM $su2AIM;fun3dAIM;cart3dAIM
```

capsIntent. The "capsIntent" Body attribute is used to disambiguate which AIM instance should receive a given Body targeted for the AIM. Bodies from the "capsAIM" selection with a matching string attribute "capsIntent" are passed to the AIM instance.

0.1.7 Creating ESP Inputs for CAPS

Creating input for use with CAPS must be a **forethought** not and afterthought! That is, if you generate an ESP geometry and then want to use different analysis AIMS on it you may have already made choices in the generation of that geometry that are not compatible with all of the AIMs of interest. To further the discussion the topic of fidelity and how it is used inside of CAPS follows:

Fidelity is the most important thing to keep in mind when generating geometry input for CAPS. There are two types of fidelity in the CAPS system.

Geometry Fidelity. This type of fidelity deals with the type of geometric topology that is being generated. For a detailed review the user is referred to EGADS documentation. For this discussion, a list of the types are given in increasing complexity as: Node, Edge, Loop (closed wire), Face, Shell (empty solid), Body (solid), Model (complex Body). In CAPS it is important to understand that the geometry construction philosophy is to only generate what is required and not attempt to reverse engineer anything.

Analysis Fidelity. This type of fidelity is more common to the everyday engineer. This has to do with analysis fidelity. For example, in aerodynamics you may have a tool that only requires wetted area, or a flat uncambered plate, or a cambered mid plane, or a water tight closed solid. These could be used to support analysis of drag, linear aerodynamics, nonlinear transpiration based calculations and or body fitted CFD such as Euler or RANS analysis. At this point reread the above paragraph on Geometry Fidelity. The conclusion is that Geometry and Analysis Fidelity are tightly linked.

After reading about the different fidelity definitions an appreciation of understanding how an ESP model will be used downstream should be forming. That is before any geometry is created the user should have an idea of what it is going to be used for. If the answer is many ways (i.e. many different analyses at multiple fidelities) then the appropriate types of geometry must be constructed to support the downstream intentions, and it can be more than one!

0.1.7.1 Design Parameters

The first step in creating geometry for CAPS at any fidelity is determining the design parameters that will drive the geometry. These parameters have may useful purposes. The first is they make you think about what you are doing. The less sarcastic reasons for them are:

Parametric Geometry Design parameters allow the model you are generating to be parametric. Many times a conceptual designer may inform you what parameters they are interested in.

Analytic Sensitivities When the geometry is created the sensitivity of that final "shape" is calculated with respect to all of the design parameters. These sensitivities may be fed to a mesh and downstream analysis to calculate engineering derivatives that are useful for design space exploration, uncertainty quantification, etc.

Ties Geometric Fidelities Together Possibly the most important reason for design parameters is that it allows the construction of multiple geometry fidelities to support multiple analyses at different fidelities to be tied together. That is all geometry is created from a single source of design parameters. This allows a seamless transition between fidelities without cumbersome reverse engineering or model regeneration.

0.1.7.2 Set Parameters

The second step is to determine how to manipulate the information defined in the design parameters into data that can be used to layout geometry. This is not always simple. For example, a *wing aspect ratio* may be a design parameter. But it does not tell you anything about how to generate geometry. However if another design parameter is *span* OR *area* then the opposite can be determined. For example:

```
# Design Parameters
DESPMTR   AspectRatio     10
DESPMTR   Area            10
# Set Parameters
SET       Span      sqrt(AspectRatio*Area)
```

This process follows until a complete geometry can be created from the design parameters. It is possible that design parameters may need to be added or removed in this process. A designer using CAPS should determine all design and set parameters before creating ANY geometry.

0.1.7.3 Geometry Primitives

Next geometry in ESP can be constructed in either top-down or bottom-up fashion. Many primitive shapes are supplied. Additionally, user has the ability to generate UDP's (see ESP documentation for information on this). These primitive shapes will be driven by the design and set parameters previously defined to generate the desired geometry. Note *primitives are not required to be solids!*

0.1.7.4 Specific AIM Attributes

Finally, if you are interested in a particular type of analysis it is important to understand what is required of its associated AIM from an attribution standpoint. You need to read that AIM's specific documentation first! Then you need to make sure that the required attributes appear in the ESP input being discussed currently.

0.1.8 Additional Resources

Documentation for CAPS is presented in this overview in addition to a detailed description of pyCAPS and each of the individual AIM applications. What follows is a description of all the available CAPS documentation. In many cases the specific application that an AIM supports is not distributed by CAPS. The AIMs themselves are freely available, however the specific applications must be obtained independently by the user. The documentation for these products discusses the AIMs interface to these tools and do not replace the individual tools documentation. AIMs may make is significantly easier to start using a new capability but they do not replace specific tool knowledge.

CAPS_Overview.pdf - *This document* provides an overview of what CAPS is and how to use it.

pdf/pyCAPS.pdf - A complete review of pyCAPS significantly expanding on what was presented in the CAPS_↩ Overview document. pyCAPS is presented in a way that enables a new user to get started and an advanced user to look up detailed information on specific functionality. This document in addition to the ESP help guide are critical to creating geometry and running CAPS.

CAPSapi.pdf - A complete and up-to-date reference documentation for CAPS API.

0.1.9 Interface Module (AIM) related Documentation

0.1.9.1 AIM Documentation: Meshing

pdf/CAPS_AIM_aflr2.pdf - Describes AIM for AFLR2, a 2D unstructured meshing software applying Advancing-↩ Front/Local-Reconnection (AFLR) procedure developed by David L Marcum [18], [17].

pdf/CAPS_AIM_aflr3.pdf - Describes AIM for AFLR3, an unstructured volume meshing software applying Advancing-Front/Local-Reconnection (AFLR) procedure developed by David L Marcum [18], [17].

pdf/CAPS_AIM_aflr4.pdf - Describes AIM for AFLR4, an unstructured 3D surface meshing software applying Advancing-Front/Local-Reconnection (AFLR) procedure developed by David L Marcum [18], [17].

pdf/CAPS_AIM_egadsTess.pdf - Describes AIM for EGADS tessellation, which corresponds to the native meshing capabilities supplied with the EGADS package. Triangular and quadrilateral [8] unstructured surface meshes may be created.

pdf/CAPS_AIM_tetgen.pdf - Describes AIM for TetGen [25], a free volume meshing technology that's library is linked to CAPS.

pdf/CAPS_AIM_delaundo.pdf - Describes AIM for Delaundo, a free 2D meshing technology that has the ability to grow boundary layer meshes for 2D, RANS calculations.

pdf/CAPS_AIM_pointwise.pdf - Describes AIM for Pointwise [15], a mesh generation software for computational field simulations.

0.1.9.2 AIM Documentation: Aerodynamic Solvers

pdf/CAPS_AIM_avl.pdf - AVL [10] is a linear subsonic aerodynamic solver developed by Mark Drela. For distribution see: http://web.mit.edu/drela/Public/web/avl/

pdf/CAPS_AIM_awave.pdf - AWAVE [20] is a wave drag estimation tool distributed by NASA.

pdf/CAPS_AIM_friction.pdf - Friction [19] is a profile and viscos drag estimation tool that is distributed with permission with CAPS.

pdf/CAPS_AIM_su2.pdf - SU2 is an open source solver [21], [22] distributed by Stanford University

pdf/CAPS_AIM_fun3d.pdf - FUN3D [2] is a NASA developed and distributed solver commonly used by U.S. government agencies and U.S. contractors.

pdf/CAPS_AIM_xfoil.pdf - XFOIL is an open source, subsonic airfoil analysis tool developed by Mark Drela. For distribution see: http://web.mit.edu/drela/Public/web/xfoil/ .

pdf/CAPS_AIM_tsfoil.pdf - TSFOIL is a transonic airfoil analysis tool available at http://www.dept.aoe.↩vt.edu/~mason/Mason_f/MRsoft.html .

pdf/CAPS_AIM_mses.pdf - This AIM can be used to interact (through file I/O) with the airfoil analysis tool MSES.

0.1.9.3 AIM Documentation: Structural Solvers

pdf/CAPS_AIM_mystran.pdf - MYSTRAN [4] is a free linear structural solver.

pdf/CAPS_AIM_nastran.pdf - NASTRAN [24] is an industry standard structural solver with nonlinear, aerodynamic and design capability.

pdf/CAPS_AIM_masstran.pdf - Masstran is a simple utility to compute mass properties from a finite element bdf mesh.

pdf/CAPS_AIM_hsm.pdf - This AIM can be used to interact with Hybrid Shell Model (HSM) [9] developed by Mark Drela at MIT

pdf/CAPS_AIM_tacs.pdf - This AIM can be used to interact via file I/O with finite element solver TACS

0.1.9.4 AIM Documentation: Design, Optimization and Systems

pdf/CAPS_AIM_fun3d.pdf - This AIM can be used to interact with FUN3D, a NASA developed and distributed solver commonly used by U.S. government agencies and U.S. contractors.

pdf/CAPS_AIM_cart3d.pdf - This AIM can be used to interact with CART3D [1],a high-fidelity inviscid analysis package for conceptual and preliminary aerodynamic design, developed by NASA

pdf/CAPS_AIM_interference.pdf - This AIM can be used to determine the interference between a collection of solid bodies.

AIMdevel.pdf - A detailed description on AIM development, including documentation of API and helper functions.

Developing new AIMs and enhancing the existing ones is an on-going activity. If you need to interface an analysis that is not listed here or need to modify an existing one to make it suitable for your workflow, please contact us for further assistance.

Bibliography

[1] M Aftomis, M Berger, and G Adomavicius. A parallel multilevel method for adaptively refined cartesian grids with embedded boundaries. Number AIAA-2008-0808, Reno, NV, Jan. 2008. American Institute of Aeronautics and Astronautics. 8

[2] Robert T. Biedron, Jan-Renee Carlson, Joseph M. Derlaga, Peter A. Gnoffo, Dana P. Hammond, William T. Jones, Bil Kleb, Elizabeth M. Lee-Rausch, Eric J. Nielsen, Michael A. Park, Christopher L. Rumsey, James L. Thomas, and William A. Wood. *FUN3D Manual: 12.7*, May 2015. 8

[3] Dean E. Bryson, Robert Haimes, and John Dannenhoffer. Toward the realization of a highly integrated, multi-disciplinary, multifidelity design environment. In *AIAA Scitech 2019 Forum*. 1

[4] William Case. *MYSTRAN General Purpose Finite Element Structural Analysis Computer Program (Linux Version 6.35) [Software Manual]*, Nov. 2011. Available from http://www.MYSTRAN.com. 8

[5] John Dannenhoffer. Opencsm: An open-source constructive solid modeler for mdao. In *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, number 2013-0701. American Institute of Aeronautics and Astronautics, Jan. 2013. 2

[6] John Dannenhoffer and Nitin Bhagat. Towards modeling for design: Using real-time collaborative environment in caps. In *AIAA Scitech 2022 Forum*. 1

[7] John Dannenhoffer and Robert Haimes. Conservative fitting for multi-disciplinary analysis. In *52nd Aerospace Sciences Meeting*, number 2014-0294. American Institute of Aeronautics and Astronautics, Jan. 2014. 4

[8] Julia Docampo-Sanchez and Robert Haimes. Towards fully regular quad mesh generation. In *AIAA Scitech 2019 Forum*. 7

[9] Mark Drela, Marshall Galbraith, Robert Haimes, Steven R. Allmaras, and David Darmofal. Hybrid shell model for aeroelastic modeling. In *AIAA Scitech 2019 Forum*. 8

[10] Mark Drela and Harold Youngren. *AVL (Athena Vortex Lattice) 3.30 User Primer*, Aug. 2010. Available from http://web.mit.edu/drela/Public/web/avl/. 8

[11] Ryan J. Durscher and Dennis Reedy. pycaps: A python interface to the computational aircraft prototype syntheses. In *AIAA Scitech 2019 Forum*. 4

[12] Robert Haimes and John Dannenhoffer. The engineering sketch pad: A solid-modeling, feature-based, web-enabled system for building parametric geometry. In *21st AIAA Computational Fluid Dynamics Conference*, number 2013-3073. American Institute of Aeronautics and Astronautics, Jun. 2013. 1

[13] Robert Haimes and Mark Drela. On the construction of aircraft conceptual geometry for high-fidelity analysis and design. In *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, number 2012-0683. American Institute of Aeronautics and Astronautics, Jan. 2012. 2

[14] Chritopher M. Heath and Justin S. Gray. OpenMDAO: Framework for Flexible Multidisciplinary Design, Analysis and Optimization Methods. In *8th AIAA Multidisciplinary Design Optimization Specialist Conference (MDO)*, pages 1–13, Honolulu, Hawaii, 2012. 1

[15] Steve L. Karman and Nicholas J. Wyman. Automatic unstructured mesh generation with geometry attribution. In *AIAA Scitech 2019 Forum*. 7

[16] Raymond M. Kolonay. A physics-based distributed collaborative design process for military aerospace vehicle development and technology assessment. *International Journal of Agile Systems and Management*, 7(3-4):242–260, 2014. 1

[17] David L Marcum. Unstructured grid generation using automatic point insertion and local reconnection. *The Handbook of Grid Generation*, pages 18–1, 1998. 7

[18] David L. Marcum and Nigel P. Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. *AIAA Journal*, 33(9):1619–1625, Sep. 1995. 7

[19] W. H. Mason. *FRICTION - Skin Friction and Form Drag Program*, Jan. 2006. Available from http://www.dept.aoe.vt.edu/ mason/Mason_f/MRsoft.html. 8

[20] L. A. McCullers. *AWAVE: User's Guide for the Revised Wave Drag Analysis Program*, Apr. 1992. 8

[21] F. Palacios, M. R. Colonno, A. C. Aranake, A. Campos, S. R. Copeland, T. D. Economon, A. K. Lonkar, T. W. Lukaczyk, T. W. R. Taylor, and J. J. Alonso. Stanford university unstructured (su2): An open-source integrated computational environment for multi-physics simulation and design. In *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, number 2013-0287. American Institute of Aeronautics and Astronautics, Jan. 2013. 8

[22] F. Palacios, T. D. Economon, A. C. Aranake, S. R. Copeland, A. K. Lonkar, T. W. Lukaczyk, D. E. Manosalvas, K. R. Naik, A. S. Padron, B. Tracey, A. Variyar, and J. J. Alonso. Stanford university unstructured (su2): Open-source analysis and design technology for turbulent flows. In *52nd Aerospace Sciences Meeting*, number 2014-0243. American Institute of Aeronautics and Astronautics, Jan. 2014. 8

[23] Phoenix Integration. http://www.phoenix-int.com/. Accessed: May 2016. 1

[24] Michael Reymond and Mark Miller. *MSC NASTRAN Quick Reference Guide Version 68*, 1996. 8

[25] Hang Si. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2):11:1–11:36, Feb. 2015. 7