AIAA SciTech Forum
23-27 January 2023, National Harbor, MD & Online
AIAA SCITECH 2023 Forum

10.2514/6.2023-0550

An Integrated Design Environment for the Engineering Sketch Pad

John F. Dannenhoffer, III[*]
Aerospace Computational Methods Laboratory
Syracuse University, Syracuse, New York, 13244

Robert Haimes[†]
Aerospace Computational Design Laboratory
Massachusetts Institute of Technology, Cambridge, Massachusetts, 02139

Over the last decade, the Engineering Sketch Pad (ESP) has been adopted in many organizations for the analysis of geometrically-complex configurations, such as aerospace vehicles. A unique strength of ESP is the analytic computation of sensitivities of the configuration and its associated tessellations, all as a function of the user-specified design parameters. Several CFD solution systems can use these sensitivities with their adjoint-based flow solutions to modify the design parameters to achieve a near-optimum design of a fixed configuration. In a sense, these systems can be used to improve a design but are not directly suitable for the evolution of a design from conceptual, to preliminary, to final design.

Described herein is an Integrated Design Environment (IDE) that allows a user to evolve a design from the earliest, simple descriptions and analyses through detailed design and analysis. First the guiding principles of the new IDE are presented, followed by a case study that illustrates the use of the IDE through the design of a rubber-powered model aircraft.

The Engineering Sketch Pad (ESP)

Over the last decade, the Engineering Sketch Pad (ESP)[1] has been adopted by many organizations as the basis for the analysis of geometrically-complex configurations, such as aerospace vehicles. ESP is a geometry creation and manipulation system whose goal is to support the analysis methods used during the design process via the Computational Aerospace Prototype Syntheses (CAPS) program.[2] ESP's user interface runs in any modern web browser and its calculations are executed in a server-based backend program.

ESP is a solid modeler, which means that the construction process guarantees that models are realizable solids, with a watertight representation that is essential for mesh generators. Since some analyses require representations in terms of sheets or wires, ESP can support those too.

ESP's models are parametric, meaning that they are defined in terms of a feature tree (which can be thought of as the "recipe" for how to construct the configuration) and a set of user-defined design parameters that can be modified to generate families of designs.

Like all feature-based systems, ESP models start with the generation of primitives, which can either be one of the standard primitives (box, sphere, cone, cylinder, torus), or can be grown from a sketch as either an extrusion, body of revolution, or a blend of a series of sketches. In addition, ESP allows users to

[*]Associate Professor, Mechanical and Aerospace Engineering, AIAA Associate Fellow.
[†]Principal Research Engineer, Department of Aeronautics & Astronautics, AIAA Member.

create their own primitives; for example, a series of airfoil generators are shipped with ESP. Primitives can be modified via transformations (translate, rotate, scale, mirror); fillets, chamfers, or hollows can also be applied. Finally, bodies can be combined via boolean-like operators such as intersect, subtract, and union.

ESP maintains a set of global and local attributes on a configuration that are persistent through rebuilds. This association is essential in the support of multi-fidelity models (wherein the attributes can be used to associate conceptually-similar parts in the various models) and multi-disciplinary models (wherein the attributes can be used to associate surface groups which share common loads and displacements). User-specified attributes are also used to mark faces and edges with information such as nominal grid spacings or material properties.

A key difference from ESP and all other available modeling systems is the ESP allows a user to compute the sensitivity of any part of a configuration with respect to any design parameter. Many of ESP's commands have been analytically "differentiated" or have used "operator overloading", making the computation of sensitivities efficient (since there is no need to re-generate the configuration) and accurate (since there is no truncation error associated with "differencing"). A few feature types still require the use of finite-differenced sensitivities, for which a new mapping technique is used to ensure robustness.

As mentioned above, ESP is extensible, in that users can add their own user-defined primitives (UDPs) and user-defined functions (UDFs), both of which are written in C, C++, or FORTRAN and are compiled, using either top-down or bottom-up process. UDPs/UDFs are coupled into ESP dynamically at run time. Additionally, a user can write a user-defined component (UDC), which can be thought of as a "macro".

ESP models are defined in `.csm` files, which are human readable ASCII files that use a CAD-traditional stack-like process, but which also allows for looping (via patterns), logical (if/then) constructs, and error recovery via thrown/caught signals.

ESP's back-end (server) runs on a wide variety of modern compute platforms, including LINUX, MAC-OS, and Windows. ESP's user-interface (client) runs in most modern web browsers, including FireFox, Google Chrome, Safari, and chromium Edge. ESP is an open-source project (using the LGPL 2.1 license) that is distributed as source, and is available from `acdl.mit.edu/ESP`.

The geometry in ESP is coupled with a wide variety on analytical tools via CAPS. Until recently, ESP (and CAPS) have been primarily used as an **analysis** environment:

- Geometry is generated via a .csm file
- One or more analyses (AIMs) are set up and executed
- The "results" are sometimes written to the screen (stdout) or sometimes written to files to possibly be examined or visualized by an external program

These analyses are typically executed in isolation, with no connection to the results of prior analyses. This is typical of the workflow associated with normal Computational Fluid Dynamics (CFD) and Finite Element Method (FEM) analyses.

The Design Problem

In contrast to analysis, **design** is a workflow or process that involves a series of interconnected models of both the form and function of the product. A typical design process can involve many users working over a long period of time, in possibly geographically-dispersed locations. Unlike analysis, which typically uses a single instance of the model, models in design evolve over time, generally with increasing detail and complexity. Also, design is not a single-pass process, but frequently involves exploration of alternatives, some of which might be abandoned, but all of which should be documented in some way. Lastly in design, it is not uncommon for the requirements to change and the customer learns more about the evolving design.

The objective of the current work is to extend ESP from geometry and analysis to a design system.

Users' Requirements

In creating any new capability, it is essential to develop a set of requirements. For an Integrated Design Environment (IDE), a survey of designers developed the following list:

- A tool that can be used by all individuals involved with the design and its process
- Have a central repository of all design data including initial mission and functional requirements
- Provide a way of including outside documents (such .pdf and .png files)
- Provide a mechanism for the user to document the how and why design decisions were made
- Maintain history of the evolving design and design decisions
- Support various optimization schemes (provide sensitivities when available)
- Support user-written workflow (Python) scripts
- Allow for connecting to MDO Frameworks
- Be robust; be able to recover from errors/aborts
- Be able to be suspended (for example, during a long-running analysis or runs supporting Design Of Experiments)
- Be able to "restart" after being suspended/aborted
- Be able to execute workflow scripts in a batch setting (production) or interactively (using a debugging analogy) without modification
- Give users methods to interactively visualize the geometry and analysis results at any stage of the design
- Allow the user to generate simple graphs from within scripts
- Allow geometric models to evolve over time
- Be able to build upon the work done in previous sessions
- Support branching, merging, and pruning of the design-decision tree
- Allow collaboration by geographically-disperse users

Guiding Principles

These requirements could be translated into the following guiding principles:

- Explicitly encode *Design Intent*
- Provide feedback to the designer/engineer to help them learn (and add value) during the design process
- Support (and reinforce) best practices
- Allow the users to be agile as they gain more experience with their design and with the system
- Provide an interactive experience that facilitates geometric construction coupled with process management
- Allow for easy *debugging* and error handling

- Facilitate collaboration

The IDE-specific guidelines are in addition to the general guidelines that have been adopted throughout the entire ESP development process; the system should:

- work for user, not the user working for the system
- be based on a mental model that is *easy* for the user to grasp
- never lose anything they have done (backward compatibility)
- solve the user's real problem, not their stated request
- be responsive to the changing needs of the users
- never surprise the user
- not require reverse engineering; and
- be tested thoroughly.

A search of any prior "art" that can be directly mimicked to assist with the software design of the IDE yielded no good results. So concepts were borrowed from commercial parametric CAD (but with rich attribution), multi-disciplinary optimization (MDO) frameworks (with their associated embedding of the process/workflow), video meetings (for geographically-dispersed collaborations), database management system, software repositories, and software Integrated Development Environments (IDEs).

New Integrated Design Environment (IDE)

In recognizing that the design workflow may require a long running and complex process involving a mix of multi-fidelity and multidisciplinary analyses, the IDE needed to give the user(s) the ability to manage the complexity of their process, be able to answer *what if* questions, continually checkpoint the state of the session, and retrieve design decisions at a later time (*why did this happen?*). The solution adopted here is to break the design execution into a series of *phases*, which can be thought of as branches of the design-decision *tree*. Typically each new phase is a built upon its parent phase, just like in various versions of a computer program builds upon its predecessors. But also like in software development, occasionally new "branches" can be created, in which a user might want to examine several alternatives.

This design of the IDE that allows for:

- incrementally building up the design workflow;
- editing and debugging the workflow;
- having a flexible and robust execution model;
- supporting multiple participants in the design session: discipline engineers, designers and managers;
- adding any number of new design tools and analysis suites;
- a common and consistent user interface throughout; and
- visualizing geometry, geometry related data, and other useful information (again for debugging)

Also, like a software IDE the user need not leave the design IDE to edit, execute, and debug the geometry construction (CSM) or workflow (python).

Demonstration

To demonstrate the new IDE, a mini-design project was executed, which was patterned after the senior design project at Syracuse University in the Spring 2022 semester. The students were asked to design, build, and fly a rubber-powered aircraft that carried golf balls. The "score" associated with an airplane was the product of the number of seconds aloft and the square of the number of balls carried.

As with all design exercises, the design process was conducted in a series of "phases". In the earliest phases, a large number of assumptions were made and the analyses were rather simple. Throughout the course of the design process, many of the earlier assumptions were replaced by analytical results associated with more detailed models.

The initial model (phase 1.1) was that of a simple rectangular wing, as shown in Figure 1. For this wing, there were only four design parameters: the wing area (S), the wing aspect ratio ($\mathcal{AR}$), wing thickness-to-chord ratio (t/c), and wing camber (m/c). Given this simple model, the task was to find the values of S, $\mathcal{AR}$, and cruise velocity (V) that maximized the score for a given number of balls (n). This was done with a full-factorial design of experiments, using the relations:

$$b = \sqrt{SA!R} \quad (1)$$

$$q = \frac{1}{2}\rho V^2 \quad (2)$$

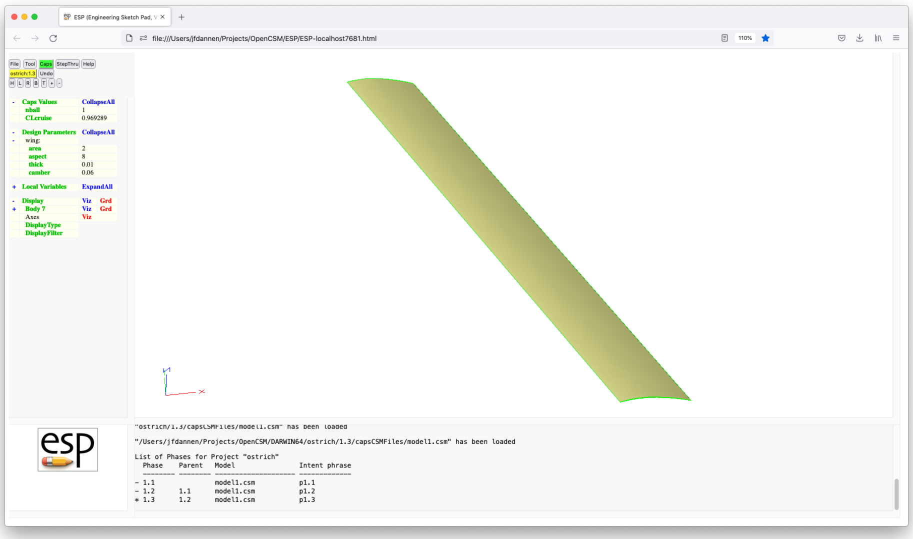
$$W = W_{\text{fixed}} + nW_{\text{ball}} + SbW_{\text{wing}} \quad (3)$$

$$C_L = W/(qS) \quad (4)$$

$$C_D = C_{D,0} + \frac{C_L^2}{\pi \mathcal{AR} e} \quad (5)$$

$$L/D = fracC_L C_D \quad (6)$$

Here, there were assumed valued for the air density (ρ), the various weight terms ($W_?$), and Oswald efficiency factor (e). All this was encapsulated in a user-defined python script that was executed directly within the IDE.


ESP (Engineering Sketch Pad, V
file:///Users/jfdannen/Projects/OpenCSM/ESP/ESP-localhost7681.html
110%
File
Tool
Caps
StepThru
Help
ostrich:1.3
Undo
H
L
R
B
T
+
-
esp

-	Caps Values	CollapseAll	
	nball	1	
	CLcruise	0.969289	
-	Design Parameters	CollapseAll	
-	wing:		
	area	2	
	aspect	8	
	thick	0.01	
	camber	0.06	
+	Local Variables	ExpandAll	
-	Display	Viz	Grd
+	Body 7	Viz	Grd
	Axes	Viz	
	DisplayType		
	DisplayFilter		

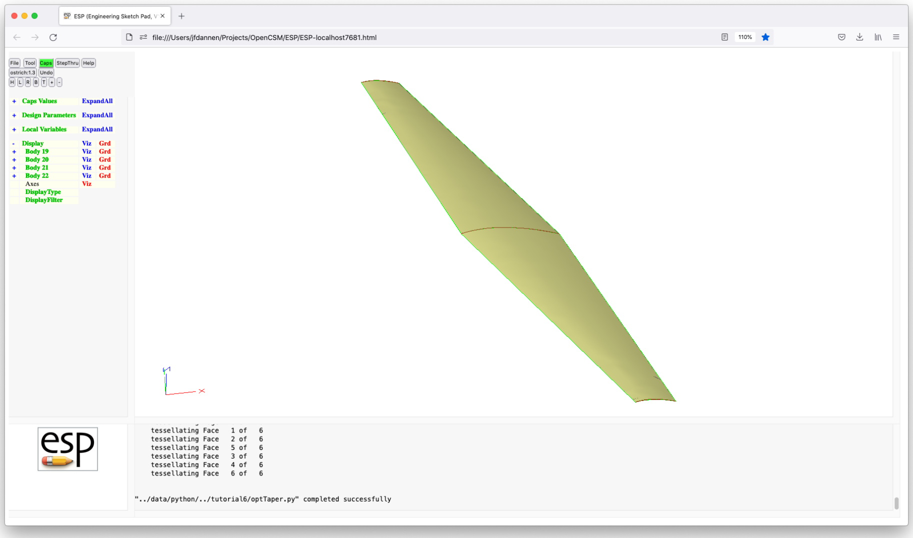
```
"ostrich/1.3/capsCSMFiles/model1.csm" has been loaded

"/Users/jfdannen/Projects/OpenCSM/DARWIN64/ostrich/1.3/capsCSMFiles/model1.csm" has been loaded

List of Phases for Project "ostrich"
  Phase    Parent   Model                Intent phrase
  -------- -------- -------------------- ------------
- 1.1               model1.csm           p1.1
- 1.2      1.1      model1.csm           p1.2
* 1.3      1.2      model1.csm           p1.3
```

Figure 1. Rectangular wing used in phase 1.1

In the next phase (1.2), the user switched from a simple rectangular wing to a tapered wing, with the addition of the taper ratio (τ). The "improved" geometrical representation is shown in Figure 2. In order to find the "optimal" taper ratio, a series of aerodynamic computations were computed via the Athena Vortex Lattice (AVL) method.[3] This program requires that the wing be represented by a series of cross-sections. Fortunately, these cross-sections are known since they were used to generate the outer mold line (OML) shown in Figure 2; the "view" of the configuration needed for AVL is shown in Figure 3. The "optimization" used here was to find the taper ratio that maximized the Oswald efficiency factor (e). This was performed in another user-supplied python script, which produced the plot shown in Figure 4. This plot, which is displayed for the user, gives the user both confidence in the result and an insight into the design trade-offs.



```
    tessellating Face   1 of   6
    tessellating Face   2 of   6
    tessellating Face   5 of   6
    tessellating Face   3 of   6
    tessellating Face   4 of   6
    tessellating Face   6 of   6

"../data/python/../tutorial6/optTaper.py" completed successfully
```

Figure 2. Tapered wing used in phase 1.2

The next phase (1.3) looks at the effect of wing bending. It is well known that minimizing weight is key to design when propulsive power is limited (such as with rubber-band power), and so the wing will be made as light as possible. This in turn means that the wing will bend significantly — a concern that must be addressed. Here, it (conservatively) assumed that the wing load is elliptical and that wing structure consists of a single spar. The geometrical model is again "improved" to allow for wing bending, and the bent shape is computed via another user-supplied python script. The resulting bent wing is shown in Figure 5.

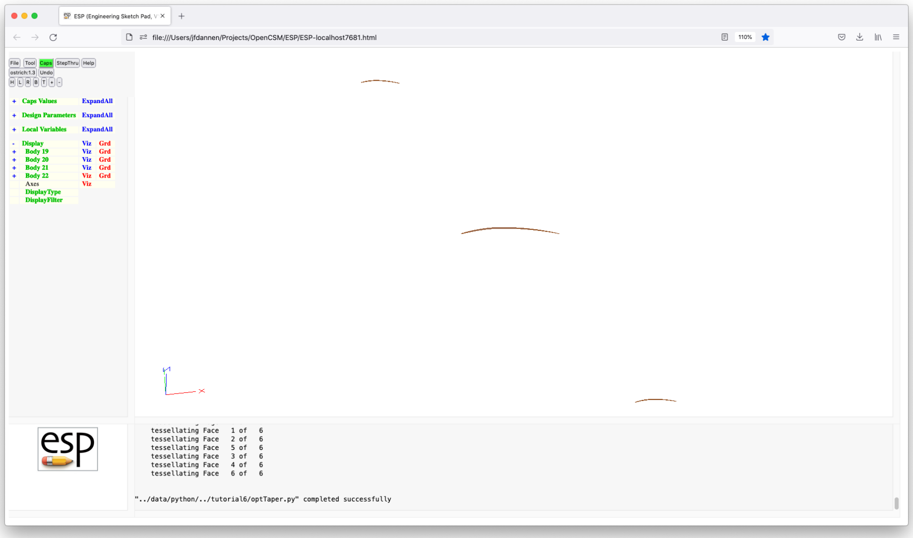
After the wing is designed, the next phase (1.4) aims to size the tail. Again this is done via trimming calculations with AVL. Figure 6 shows the cross-sections used in AVL for the trimming calculations.

As one can see, the new ESP IDE give the user the ability to enrich the geometric model as the design progresses, based upon the results of both back-of-the-envelope and CFD/FEM-based methods.

Once the above has been completed, the design team may want to explore the effect of changing the number of golf balls. To do this, the team might create a new "branch" of the design tree. This new phase, called phase 2.1, is branched off the completed phase 1.1 done previously.

At any time, the team can get a view of the phases, such as:

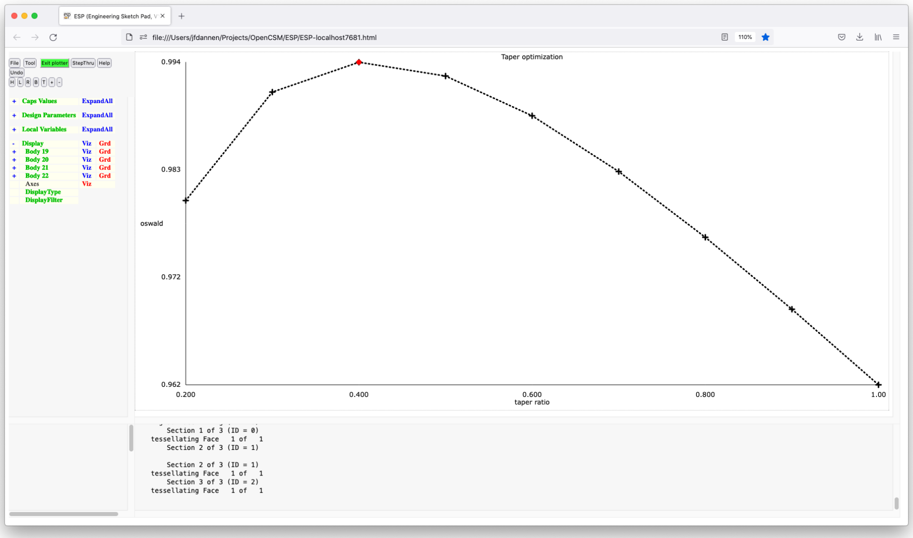
Phase	parent	description
1.1	-	baseline rectangular wing
1.2	1.1	optimal taper for one golf ball
1.3	1.2	effect of wing bending
2.1	1.1	optimal taper and wing bending for two golf balls



```
    tessellating Face   1 of   6
    tessellating Face   2 of   6
    tessellating Face   5 of   6
    tessellating Face   3 of   6
    tessellating Face   4 of   6
    tessellating Face   6 of   6


"../data/python/../tutorial6/optTaper.py" completed successfully
```

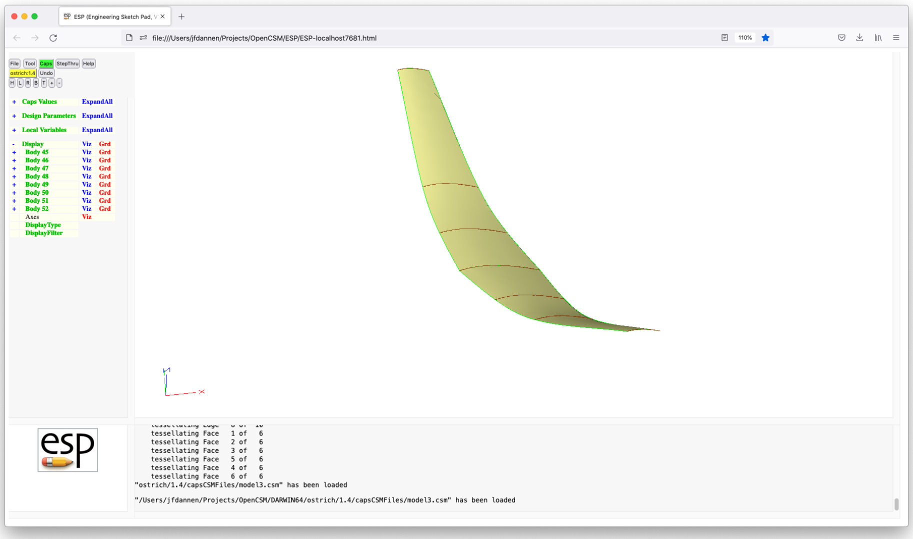
Figure 3. AVL representation of tapered wing used in phase 1.2



```
       Section 1 of 3 (ID = 0)
    tessellating Face   1 of   1
       Section 2 of 3 (ID = 1)

       Section 2 of 3 (ID = 1)
    tessellating Face   1 of   1
       Section 3 of 3 (ID = 2)
    tessellating Face   1 of   1
```

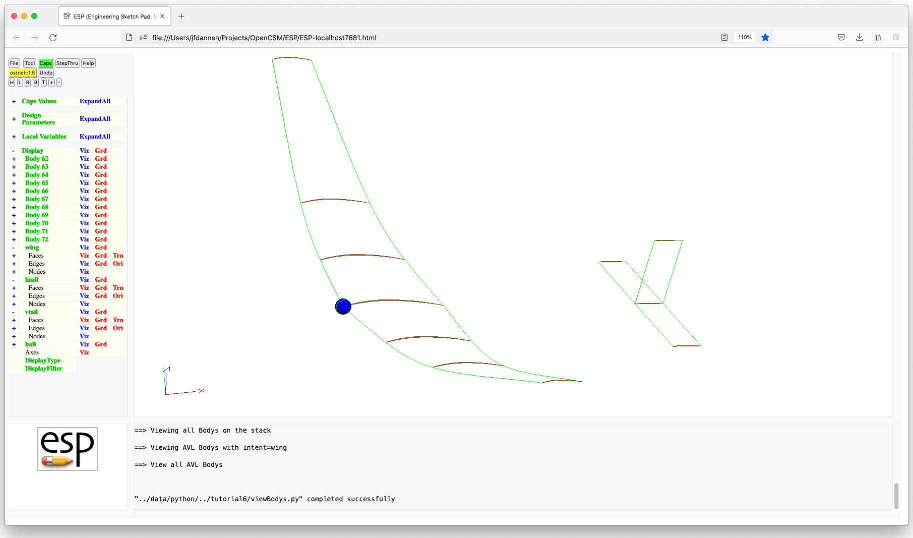
Figure 4. Oswald efficiency (e) as a function of taper ratio (τ) for the optimization of taper ratio in phase 1.2


file:///Users/jfdannen/Projects/OpenCSM/ESP/ESP-localhost7681.html
Caps Values
Design Parameters
Local Variables
Display
esp

```
    tessellating Face   1 of   6
    tessellating Face   2 of   6
    tessellating Face   3 of   6
    tessellating Face   5 of   6
    tessellating Face   4 of   6
    tessellating Face   6 of   6
"ostrich/1.4/capsCSMFiles/model3.csm" has been loaded

"/Users/jfdannen/Projects/OpenCSM/DARWIN64/ostrich/1.4/capsCSMFiles/model3.csm" has been loaded
```

Figure 5. Bent wing shape in phase 1.3



```
==> Viewing all Bodys on the stack

==> Viewing AVL Bodys with intent=wing

==> View all AVL Bodys


"../data/python/../tutorial6/viewBodys.py" completed successfully
```

Figure 6. AVL model of wing/tail used to size the tail in phase 1.4

Also, the user may examine the changes to any design input or output parameter over the various phases.

These capabilities are embedded in the latest version of the Engineering Sketch Pad (`ESP`), which is freely available at `acdl.mit.edu/ESP`.

Acknowledgment

This work was supported by the EnCAPS Project (AFRL Contract FA8650-20-2-2002): "EnCAPS: Enhanced Computational Prototype Syntheses", with Ryan Durscher as the Technical Monitor.

References

[1]Haimes, R. and Dannenhoffer, J.F., "The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry", AIAA-2013-3073, June 2013.

[2]Bryson, D.E., Haimes, R., and Dannenhoffer, J.F., "Toward the Realization of a Highly Integrated, Multidisciplinary, Multifidelity Design Environment", AIAA-2019-2225, January 2019.

[3]Drela, M. and Youngren, H., "AVL 3.40 User Primer", `https://web.mit.edu/drela/Public/web/avl/avl_doc.txt`.