AIAA Aviation Forum and ASCEND co-located Conference Proceedings
29 July - 2 August 2024, Las Vegas, Nevada
AIAA AVIATION FORUM AND ASCEND 2024
10.2514/6.2024-4304

Using OpenVSP Models in the Engineering Sketch Pad

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

OpenVSP is a parametric aircraft geometry tool that allows the user to create a 3D model of an aircraft defined by common engineering parameters. It has been used by professionals and hobbyists to create models of over 100 different aircraft types. OpenVSP models can be processed into formats suitable for engineering analysis.

The Engineering Sketch Pad (ESP) is a CAD-like system for the generation of geometric models for the analysis and design of complex configurations, such as aircraft. It is a feature-based, parametric solid modeler that has been coupled directly to a large number of fluid-thermal-structural-control analysis programs via the Computational Aerospace Prototype Synthesis (CAPS) system.

The objective of this current research is to be able to use models created in OpenVSP directly in ESP, and subsequently in CAPS. This coupling allows ESP to directly drive OpenVSP's user parameters and to generate sensitivities of the configuration with respect to changes in the user parameters.

`OpenVSP`

Since the early 1990's, `OpenVSP`[1–3] and its predecessors have been under active development for NASA. Through numerous publications,[4] its "Ground School", and annual workshops (all or which can be found online[5]), `OpenVSP` has gained widespread adoption and has made a significant impact on the aerospace industry's ability to rapidly sketch aircraft. `OpenVSP` is an open-source project that allows one to download current and past versions for several operating systems from its website.[5]

One builds an `OpenVSP` configuration by creating one or more pods, fuselages, wings, stacks, ellipsoids, bodies of revolution, humans, props, hinges, or conformals. Each of these components are defined in terms of "sensible" engineering parameters; the wing alone has about 100 user-adjustable parameters such as span, area, sweep, twist, and thickness/chord and camber (at several spanwise locations). A particular strength of `OpenVSP` is its ability to provide nearly instantaneous feedback to the user as any of the parameters associated with the components are modified, either via numerical input or via sliders. This makes it ideal for creating 3D models from 3-view drawings of aircraft.

Traditionally `OpenVSP` models are represented by a series of components that "fly in formation". Methods of combining these components, such as the use of Boolean operations, have been somewhat limited until recently. Even with these new Boolean techniques, `OpenVSP` lacks the tools necessary to add wing/fuselage fillets, for example.

Fig. 1 shows an example of the `OpenVSP` user interface when building a parametrically-defined transport configuration. Shown in its various windows are a view of the aircraft model, a list of the components in the model, a panel to edit the parameters associated with the wing, a list of the linkages between parameters, and a window that allows a user to modify any of the user parameters.

*Associate Professor, Mechanical and Aerospace Engineering, AIAA Associate Fellow.

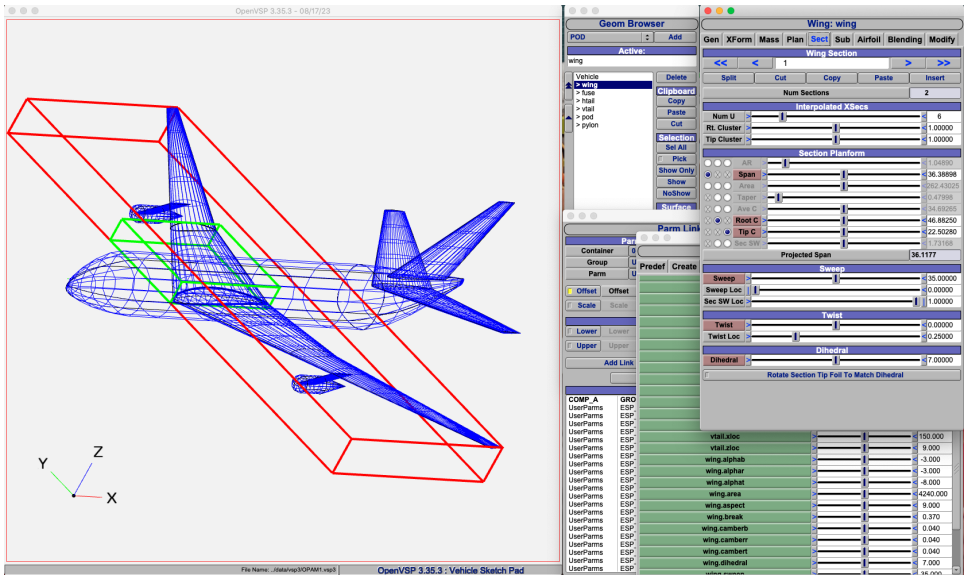

Figure 1. Some of the OpenVSP windows for the transport configuration.

The Engineering Sketch Pad (ESP)

The Engineering Sketch Pad (ESP) and it supporting components[6–8] 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.[9] Numerous publications on the effective use of ESP for a wide variety of aerospace applications can be found at its website.[10] ESP is an open-source project that allows one to download current and past versions for several operating systems from its website.[11]

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. 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 (DESPMTRs) that can be modified to generate families of designs. In many ways, ESP is like a traditional CAD system, but one focused on engineering models for analysis and design rather than manufacturing.

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.

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 user interface runs in any modern web browser and its calculations are executed in a server-based backend program. 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`.

Aircraft Design Process

Traditionally, the earliest phase of any aircraft design activity starts with a hand sketch of a proposed aircraft. Several such hand sketches are shown as examples in Raymer's classic aircraft design textbook.[12] Having such a sketch anchors a design, ensuring that the various subsequent calculations are consistent and that the size and proportion of the various aircraft components "make sense" together.

OpenVSP is a computerized implementation of hand sketches, and as such is often the starting point for modern aircraft designs. Since the purpose of its sketches is to convey overall views of an aircraft, details as to how the various components actually interface are often omitted. This omission makes downstream analyses (such as computational fluid dynamics (CFD) and finite element methods (FEM)) difficult.

ESP, through CAPS, was designed to transform an engineer's design ideas into CFD- or FEM-ready models for more detailed analysis and design. But ESP uses a somewhat CAD-like approach to creating models, making its use in the earliest design phases somewhat heavy handed.

The purpose of this research is to import OpenVSP-created models into ESP and CAPS.

Importing OpenVSP models into ESP

A user-defined primitive (UDP) has been written for ESP that allows a user to directly import OpenVSP models. This primitive generates in ESP a series of bodies, one for each component in the OpenVSP model. When the transport configuration shown in Fig. 1 is imported into ESP with the statement:

```
UDPRIM vsp3 filename $myfile.vsp3
```

it produces the bodies shown in Fig. 2. Notice on the left side of the figure that the 10 components in the OpenVSP model come into ESP as 10 bodies (listed under the word "Display").

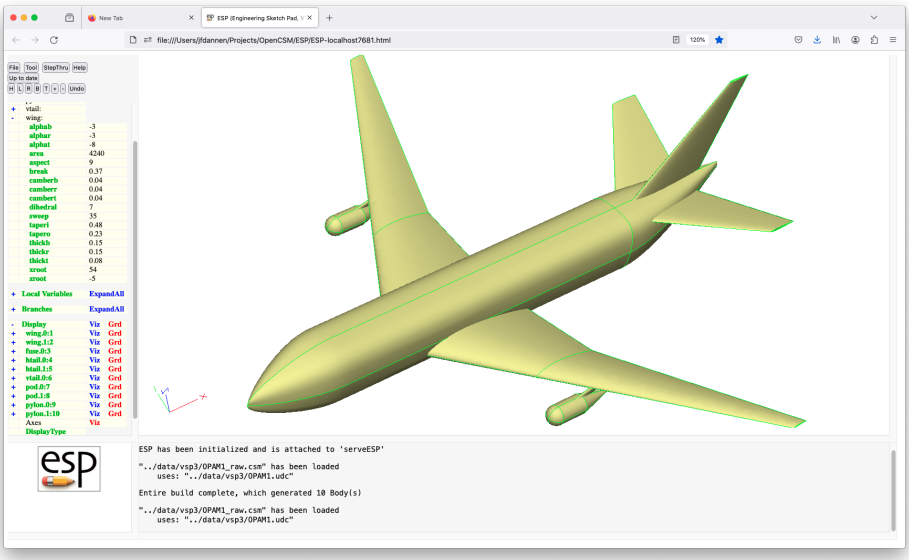

Figure 2. The transport configuration imported into ESP as 10 bodies "flying in formation".

The process has been tested on over 30 OpenVSP models (mostly from the OpenVSP Hanger); Figs. 3–5 show three examples.

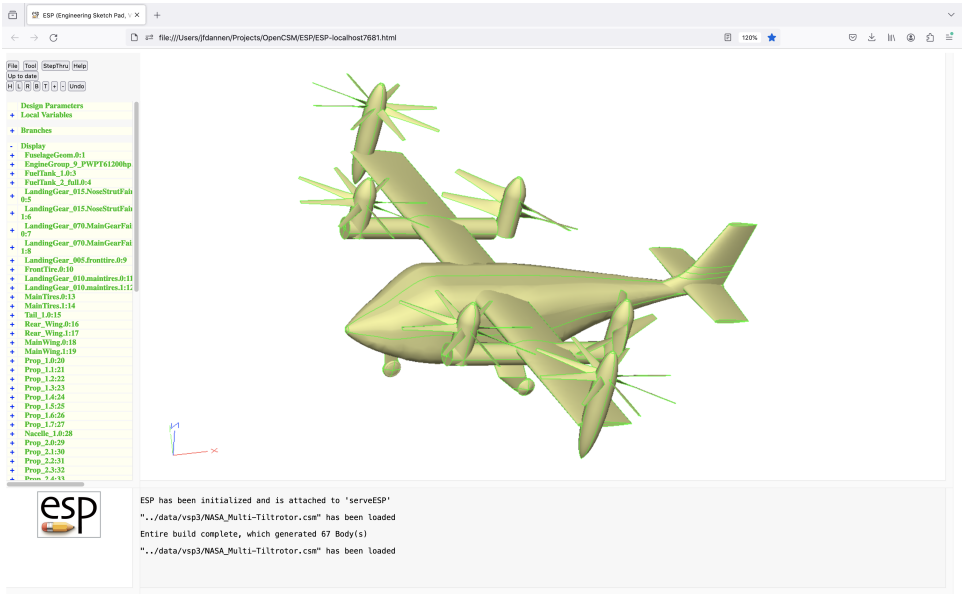

Figure 3. NASA_Multi-tiltrotor configuration.

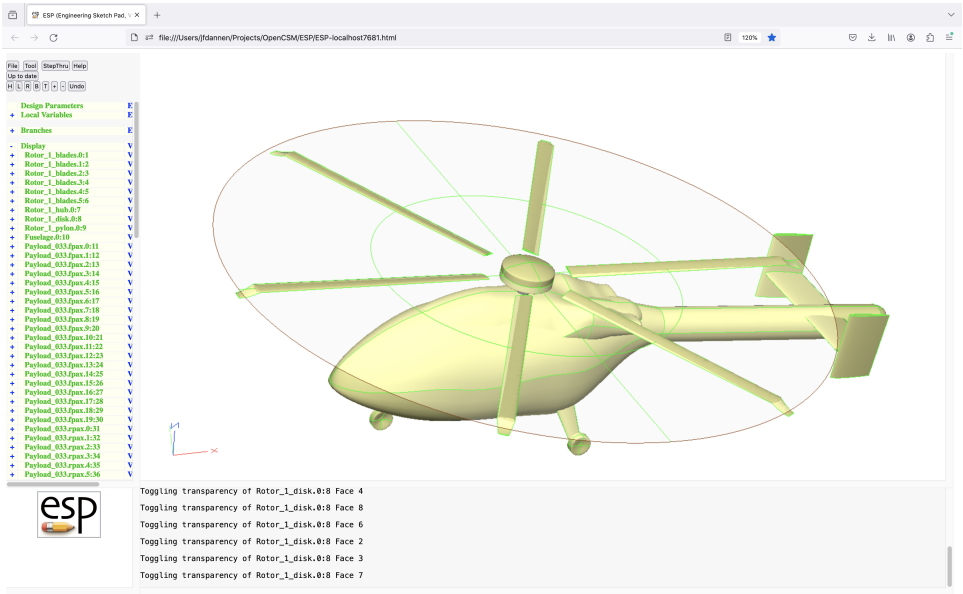

Figure 4. QEU2023-SMR-6pax-turbo-notar_450fps configuration.

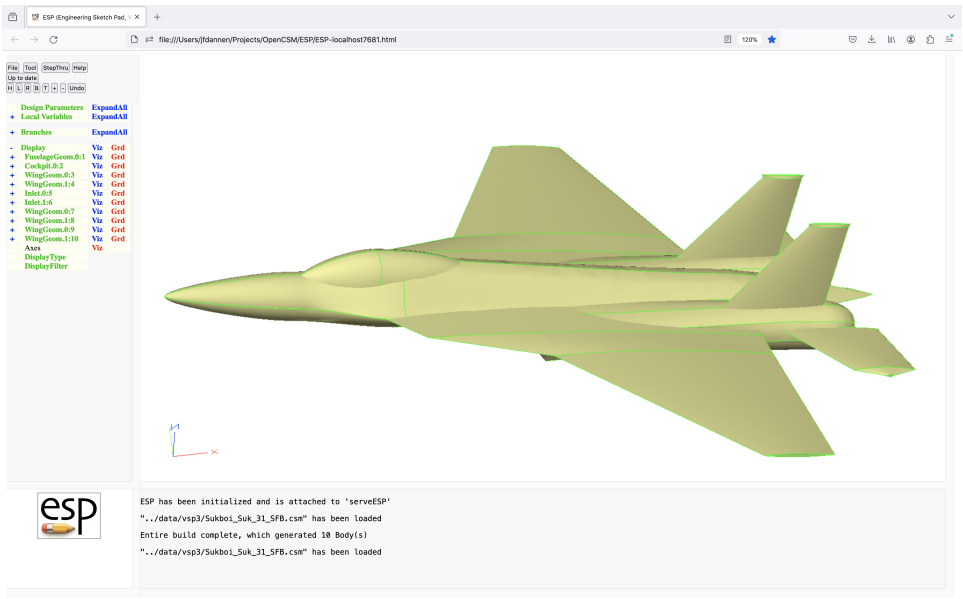

Figure 5. Sukboi_Suk_31_SFB configuration.

In developing the new method, the first major decision was how to implement the `OpenVSP`-compatible geometry into `ESP`. The initial attempt at doing this involved reproducing `OpenVSP`'s geometry creation in `ESP`. This had several important drawbacks. First, the complexity of doing this was overwhelming, especially due to the large number of inputs available for each of the components. And second, there would be no way of ensuring that the geometry creation would remain consistent as `OpenVSP` and its methods evolved. Hence it was decided to use `OpenVSP` directly to create the geometry; this could be done by having `OpenVSP` export its curves and surfaces.

The second major decision was how to allow `ESP` to modify the various `OpenVSP` user parameters (UserParms) in a model; doing such would be essential to using the `OpenVSP-ESP` link in analysis and design. Fortunately this can be done via `OpenVSP`'s scripting language, angelscript.[13] More specifically, every `ESP` DESPMTR (such as `wing:sweep`) is mapped to an `OpenVSP` UserParm in the `ESP_Group` (such as `wing.sweep`). Note that `ESP` uses the colon (:) to organize DESPMTRs whereas `OpenVSP` uses the period (.) to organize UserParms; this translation is done automatically in `ESP`.

The mechanism used to import an `OpenVSP` model consists of the following steps:

1. `ESP` creates an angelscript for `OpenVSP`'s `vspscript` that:
   - reads the `.vsp3` file;
   - sends all of `ESP`'s design parameters (DESPMTRs) to `OpenVSP` (some of which may not be in the `OpenVSP` model and are thus rejected);
   - sets the various `OpenVSP` internal parameters such that its output STEP file has the appropriate format and surface labels; and
   - exports it surfaces (and curves) in a `.stp` file.
2. `vspscript` is executed via a call to the `system()` function in C. The `VSP3_ROOT` environment variable is used to select the appropriate version of `OpenVSP`; and
3. the `.stp` file that `vspscript` wrote is read into `ESP`

When ESP reads the .stp file, it consists of one body for each (untrimmed) surface in the whole configuration. Fortunately OpenVSP's naming convention allows ESP to assemble many .stp bodies that comprise a component (such as fuselage) into a single ESP body. Hence each OpenVSP component is a separate ESP body after the UDPRIM vsp3 statement.

Since analyses of aircraft seldom want to analyzes bodies "flying in formation", one can simply add the ESP command:

```
UNION 1
```

to UNION all components into a single body, as shown in Fig. 6.

For design trade-offs, the ESP user can then change any DESPMTR; for example, changing the wing:sweep to 15°, yielding the result in Fig. 7.

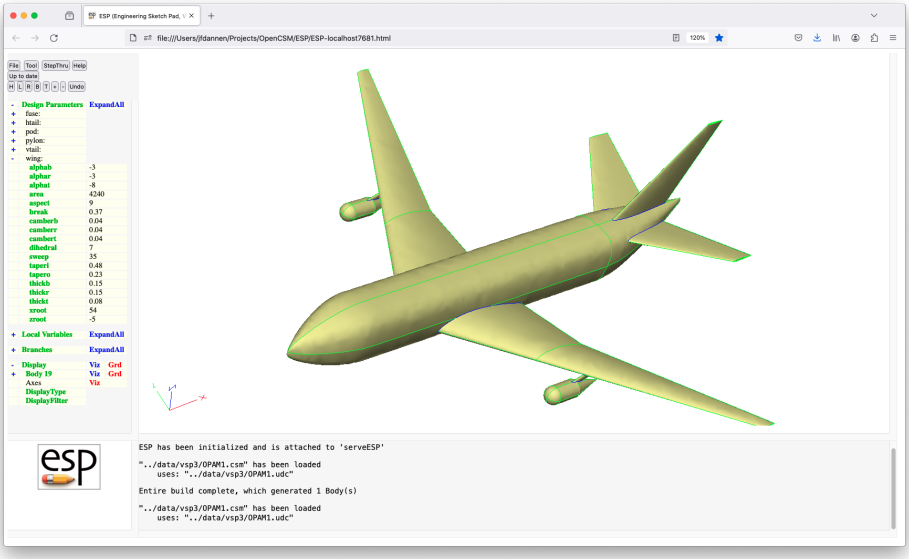

Figure 6. The transport configuration imported into ESP and UNIONed into one body.

In addition, one can easily add the wing/fuselage fillets, producing the configuration in Fig. 8.

Sensitivities

One of the unique powers of ESP is its ability to compute the sensitivity of a configuration with respect to any of the DESPMTRs.[14] This capability is essential for "completing-the-adjoint-loop" in gradient-based design methods.

Most of ESP's geometry-creating commands have been analytically differentiated (or computed directly via operator-overloading), giving sensitivities directly (that is, not via finite differences); this makes sensitivities efficient and accurate. Note that finite differences are available for those commands for which the geometric algorithm is unknown, such as fillets.

The problem with using this analytic approach with OpenVSP geometry is that one would have to modify the OpenVSP source code in hundreds of places, which would be both difficult and prone to errors.

One other complication here is that one needs the compute the sensitivity at any location on the configuration. Hence if the sensitivities were to be computed directly in OpenVSP, there would need to be a mechanism to tell OpenVSP the locations the user required.

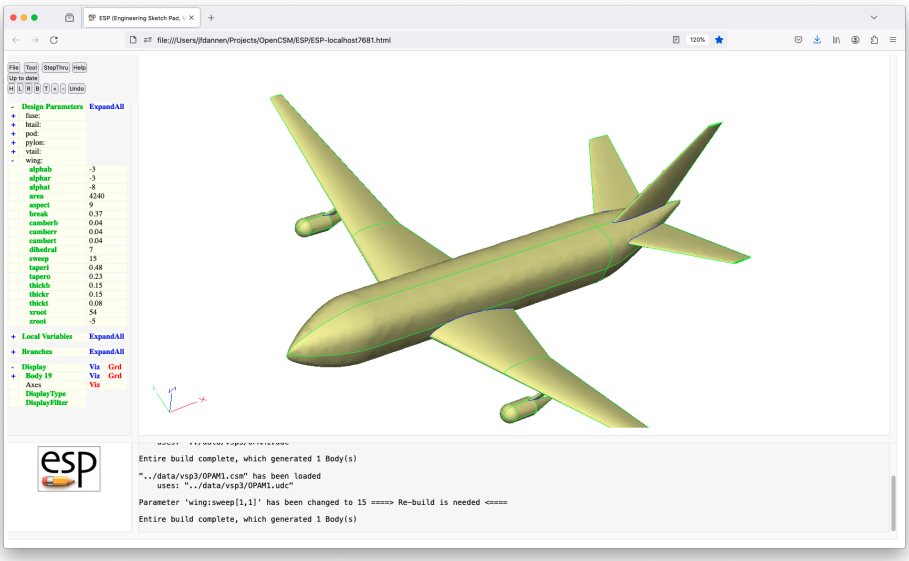

Figure 7. The transport configuration imported into ESP with sweep angle changes to 15°.

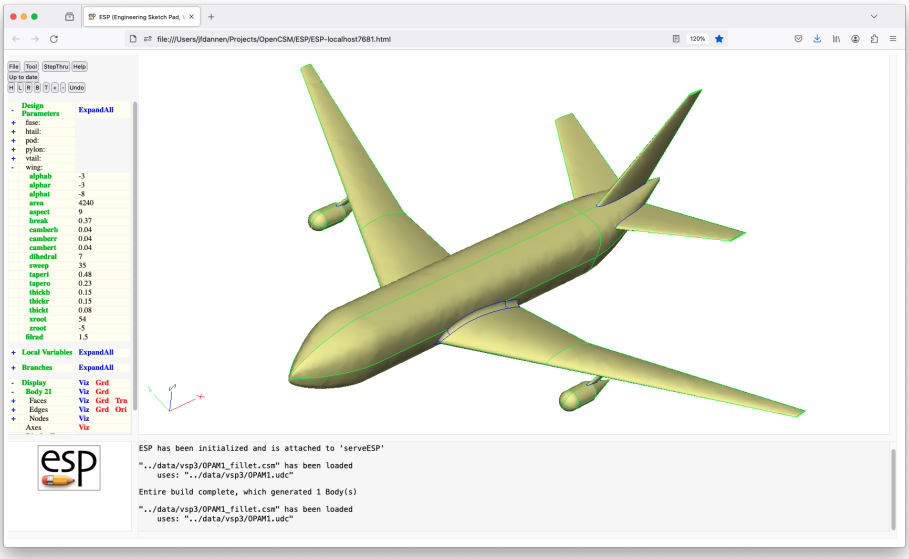

Figure 8. The transport configuration imported into ESP with fillet added at wing/fuselage junction.

Instead, here a hybrid approach is taken. First, `OpenVSP` is used to compute the sensitivity of the surface control point locations (which are contained in the `.stp` file) via finite differences. This is done by executing `OpenVSP` a second time, where one or more of its UserParms is/are perturbed and the differences in the control point locations is noted. Second, `ESP`'s internal "dot" functions are used to compute sensitivities where needed, based upon the control point sensitivities. With this approach, `OpenVSP` only needs to be re-executed when sensitivities are required for a different (set of) DESPMTRs.

Fig. 9 shows the geometric sensitivity with respect to the `fuse:midWidth`. Here red indicates that the surface grows outward whereas blue indicates that the surface grows inward (which does not occur in this case). Note that the geometric sensitivity shows only the magnitude of the surface motion that is normal to the surface. Since the fuselage width does not affect the wing, the sensitivities on the wing are zero; the discontinuity in the sensitivity at the wing/fuselage junction is a consequence.

Fig. 10 shows the geometric sensitivity with respect to the `wing:dihedral`; note that as the dihedral increases, the actual pods also moves upward. If you look closely, you can see a little bit of blue on the under-side of the pod, meaning that the surface is moving inward (that is, upward).

Finally, Fig .11 shows the tessellation sensitivity with respect to the `wing:dihedral`. The little tufts show a motion of all the tessellation points. Tessellation sensitivities are continuous at all Edges in the configuration.

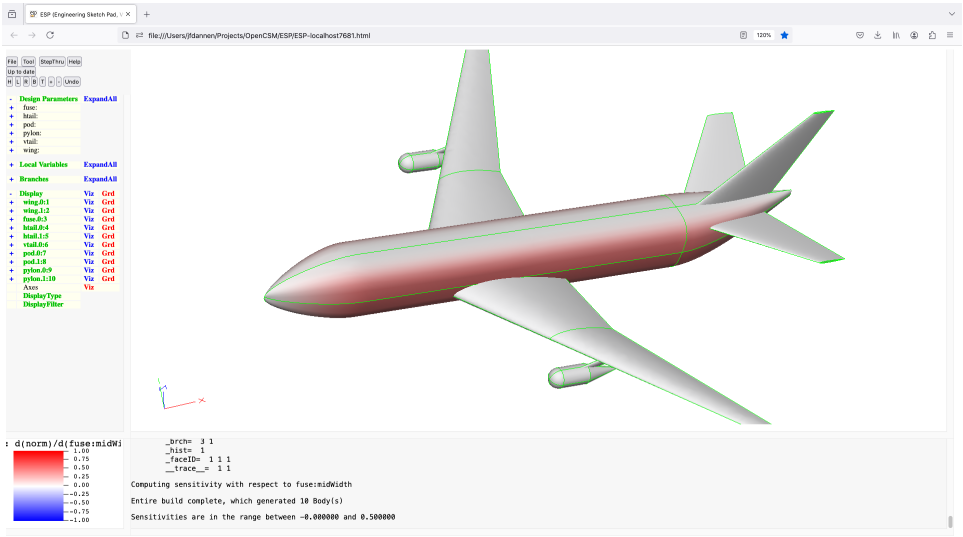

Figure 9. Geometric sensitivities with respect to `fuse:midWidth` on transport configuration.

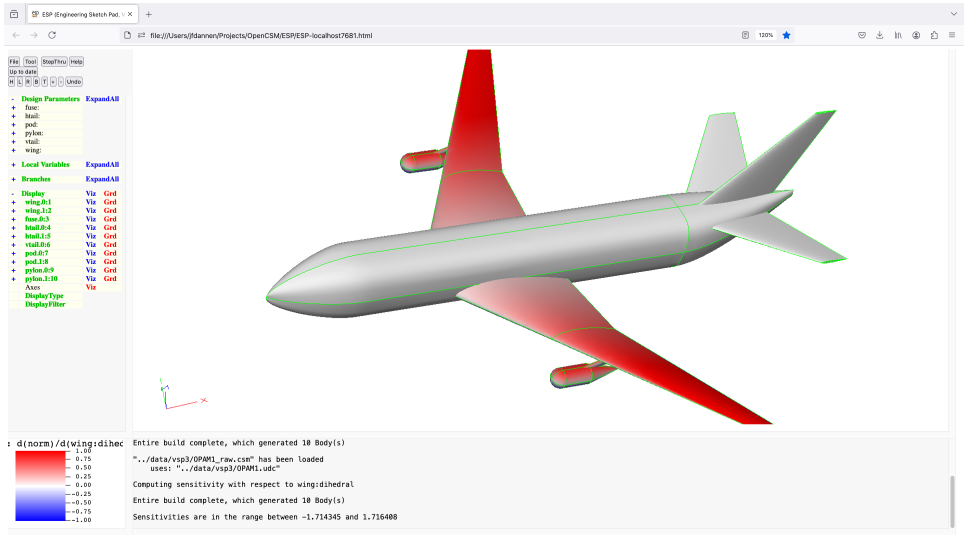

Figure 10. Geometric sensitivities with respect to `wing:dihedral` on transport configuration.

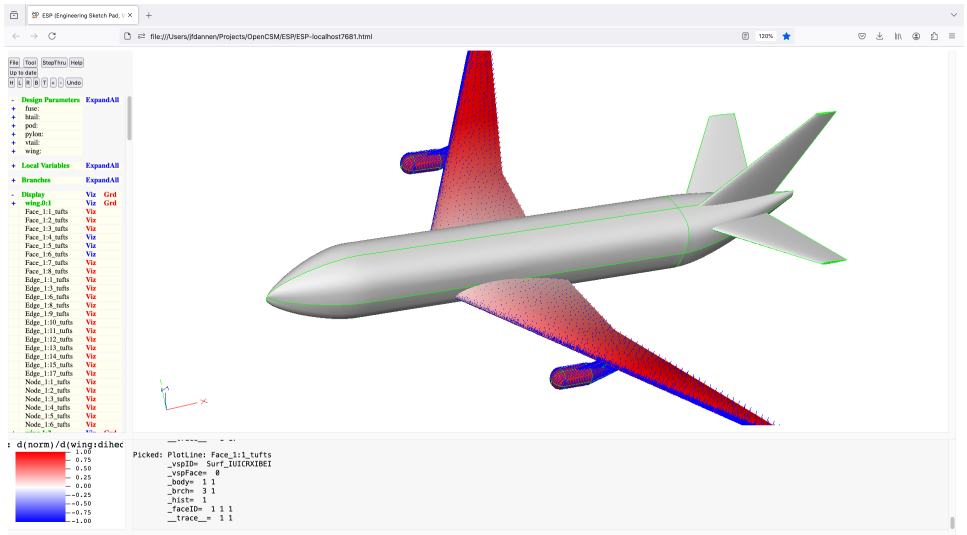

Figure 11. Tessellation sensitivities with respect to `wing:dihedral` on transport configuration.

Giving ESP access to OpenVSP's user parameters

One point glossed over in the above is how ESP gets access to OpenVSP's user parameters. This is done via the VspSetup tool within ESP. When this tool is executed, the user is asked for the name of the .vsp3 file. The tool then extracts from the OpenVSP model (via another automatically-generated angelscript) all of its user parameters (UserParms) in the ESP_Group and puts them into a ESP user-defined component (UDC). The first few lines of the UDC for the transport configuration are:

```
# generated by VspSetup from ../data/vsp3/OPAM1.vsp3

INTERFACE . ALL

   DESPMTR   fuse:aftHeight    3
   LBOUND    fuse:aftHeight    -100000
   UBOUND    fuse:aftHeight    100000

   DESPMTR   fuse:aftLength    44
   LBOUND    fuse:aftLength    -100000
   UBOUND    fuse:aftLength    100000

   DESPMTR   fuse:aftWidth    1
   LBOUND    fuse:aftWidth    -100000
   UBOUND    fuse:aftWidth    100000
```

These lines tell ESP the current value of, and lower and upper bounds for the user parameters named fuse:aftHeight, fuse:aftLength, and fuse:aftWidth. The rest of the UDC gives the values for the other user parameters. If you look on the left of Fig. 2 you will see the user parameters listed (for the wing) under "Design Parameters".

Note that the VspSetup tool only needs to be executed once for each .vsp3 file.

Summary

A link between OpenVSP and ESP has been built which allows any OpenVSP model to be imported into ESP. The user parameters in the OpenVSP model are exposed in ESP, thereby allowing the ESP user to create various instances of the model, which is useful in a design optimization environment. The ability for ESP to compute sensitivities of the OpenVSP model makes gradient-based optimization system work effectively.

Acknowledgment

This work was supported by the EnCAPS Project (AFRL Contract FA8650-20-2-2002): "EnCAPS: Enhanced Computational Prototype Syntheses", with Richard Snyder as the Technical Monitor. The author would like to thank Rob McDonald for the many conversations and insights that it took to make this project successful.

References

[1]Gloudemans, J.R., Davis, P.C., and Gelhausen, P.A., "A Rapid Geometry Modeler for Conceptual Aircraft", AIAA-1996-52, January 1996.

[2]Hahn, A., "Vehicle Sketch Pad: Parametric Geometry for Conceptual Aircraft Design", AIAA-2010-657, January 2010.

[3]McDonald, R.A. and James R. Gloudemans, J.R., "Open Vehicle Sketch Pad: An Open Source Parametric Geometry and Analysis Tool for Conceptual Aircraft Design", AIAA-2022-0004, January 2022.

[4]https://openvsp.org/wiki/doku.php?id-papers

[5]https://openvsp.org

[6]Haimes, R., and Drela, M., "On the Construction of Aircraft Conceptual Geometry for High-Fidelity Analysis and Design", AIAA-2012-0683, January 2012.

[7]Dannenhoffer, J.F., "OpenCSM: An Open-Source Constructive Solid Modeler for MDAO", AIAA-2013-0701, January 2013.

[8]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.

[9]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.

[10]`https://acdl.mit.edu/ESP/Publications`

[11]`https://acdl.mit.edu/ESP`

[12]Raymer, D., "Aircraft Design: A Conceptual Approach, Sixth Edition", ISBN: 978-1-62410-490-9, September 2018.

[13]`http://www.angelscode.com/angelscript`

[14]Dannenhoffer, J.F., and Haimes, R., "Design Sensitivity Calculations Directly on CAD-based Geometry", AIAA-2015-1370, January 2015.