AIAA AVIATION Forum
12-16 June 2023, San Diego, CA and Online
AIAA AVIATION 2023 Forum

10.2514/6.2023-3599

On Managing Geometric Models for Multi-component, Multi-disciplinary Analysis and Design

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

Nitin Bhagat†
University of Dayton Research Institute, Dayton, Ohio, 45469

Geometric models are central to the analysis and design of complex configurations, such as aerospace vehicles. As models expand to include more of the vehicle components, and to include more than one discipline, they become very complex and hard to manage. This makes understanding the linkages between components for a single discipline, or understanding the linkages between the various discipline analysis for a single component, very difficult. This problem is compounded further when one realizes that the design process involves a series of sub-models (for any component and/or discipline) that evolve over time, in which the design is changed or fidelity is enhanced; these various versions must also be managed.

Described herein is a new management scheme that directly attacks this problem. It centers around a set of user-defined component files which define the geometric models of the components that are needed by various analyses required for design. After describing the basic ideas of the new management scheme, it is demonstrated on a transport configuration. Then tools that a user can employ to understand a model are described. All this is brought together in an exercise that converts a legacy (dusty-deck) model into the new management scheme.

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

*Associate Professor, Mechanical and Aerospace Engineering, AIAA Associate Fellow.
†Senior Research Engineer, Applied Mechanics Division, AIAA Senior Member.

extrusion, body of revolution, or a blend of a series of sketches. In addition, ESP allows users to 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) are by applying fillets, chamfers, or hollows. 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, edges, and nodes with information such as nominal grid spacings or material properties.

A key difference from ESP and all other available modeling systems is that 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.

Multi-X Models (Views)

During the design of an aircraft, various coupled models are needed, representing different disciplines (such as aerodynamics, structures, thermal, controls, ...) and different fidelities (such as in conceptual design, preliminary design, and detailed design). These various models are called "views" herein. Although these various views are distinct, there needs to be communication between them.

One of the strengths of ESP is that it can build and manage multiple "views" of a single configuration, each tailored to a specific analysis method. To be effective, the views need to be driven by a single set of Design Parameters. Likewise, it is important that the views are attributed so that "common" features could be linked together.

The Table of Contents

With a multi-view model of a complex configuration, it is not uncommon that the script used to build and manage the models can get large; the script(s) for the transport configuration that will be shown in next section are more than 3000 lines long. Clearly writing a model that is understandable is a challenging task.

The first tool for managing such a large model is the creation of a table of contents of the various components and associated views. Figure 1 shows the (automatically-created) table of contents for a transport configuration. The various rows in the table correspond to the various components, while the columns show the various available views:

- **bem** is a built-up element model, which is a structural model comprised of shells (as shown in Figure 2) for use in a tool such as ASTROS[3] or NASTRAN;[4]

- **cfdInviscid** is a model of the outer mold line (OML) and a far-field box, with the control surfaces cut so that there is putty filling the various gaps (as shown in Figure 3), for use in a tool such as `SU2`[5] or `FUN3D`;[6]

- **cfdViscous** is a model of the outer mold line (OML) and a far-field box, with the control surfaces cut so that the control surfaces fly in formation with the aircraft (giving the necessary gaps around the controls) (as shown in Figure 4), for use in a tool such as `SU2` or `FUN3D`;

- **concept** is view of the layout of the various components, which is useful while the model is being built and to communicate with the various (human) stakeholders (as shown in Figures 5 and 6); and

- **vlm** contains the various "cuts" through the lifting surfaces that are needed for vortex-lattice calculations (as shown in Figure 7), for use in a tool such as `AVL`.[7]

Clearly with so many views and components, there needs to be a clear set of interfaces amongst the models. In `ESP`, all these interfaces are managed with the use of attributes on the sub-model itself, or any of its constitutive Nodes, Edges, or Faces. For example, with appropriate attributes, it is easy to identify collections of Faces on two different discipline sub-models that can be used to transfer loads or displacements between them. Attributes on various component sub-models can be used to define their geometric interfaces. And because attributes in `ESP` are persistent, linking parts of the model over various versions is done automatically.

Organizing with User-Defined Components (UDCs)

`ESP` has a macro-like capability, called used-defined components (UDCs), through which a large file can be broken into pieces. The use of UDCs will be central to managing the complexity of the overall model.

High-level UDCs

The top level UDCs are used to control the overall process. For the transport, these include:

- `transport.csm` — definition of various "views";

- `transport_init` — definition of various components.

Next, there are UDC that are responsible for creating one of the views. These build specific geometric models that are suitable for a specific analysis program. For the transport, these include:

- `viewConcept` — conceptual view, useful to understand interactions;

- `viewVlm` — for a vortex lattice method;

- `viewCfdInviscid` — outer-model line, including deflected controls and farfield boundaries for CFD analyses;

- `viewCfdViscous` — outer-model line, including free-flying controls and farfield boundaries, for CFD analyses; and

- `viewBem` — built-up element model, for use by structural solvers.

An example of the "concept" view is given in Figure 8. It simply gathers up the models (bodies) made in other UDCs.

UDCs for each Component

There are three types of UDCs in this category. This set is written for each component. For the wing component of the transport, these include:

- Initialization
  - `wingPmtrs` — definition of configuration parameters (`CFGPMTR`s) and design parameters (`DESPMTR`s)
  - `wingCalc` — high-level values computed from the CFGPMTRs and DESPMTRs
- Primitives — lowest-level geometries
  - `wingOml` — outer mold line
  - `wingWaffle` — arrangement of spars and ribs
  - `wingHinges` — location of hinge lines for control surfaces (such as ailerons and flaps)
- Models — geometric models, created by various combinations of the primitives
  - `wingVlm` — cross-sectional cuts
  - `wingCfdInviscid` — outer mold line, including deflected control surfaces
  - `wingCfdViscous` — outer mold line, including free-flying control surfaces
  - `wingBem` — built-up element model, built by intersecting a waffle with the wing shape

All of these UDCs are written as include-type UDCs, meaning that they act in a similar manner to `#include` files in the C pre-processor. An example of `wingPmtrs` is given in Figure 9; it contains only `DIMENSION`, `CFGPMTR`, and `DESPMTR` statements. An example of `wingCalcs` (see Figure 10) contains only `OUTPMTR` and `SET` statements (for values that are useful by other components).

The rules for other component-related UDCs include that they are written as an include-type UDC (`INTERFACE . ALL`), they should return immediately if the Body already exists, they should build all subordinate models and primitives, they should put all Bodys into one STOREd Group (with the name matching the model or primitive name), and they should leave the stack the same as it was upon entry. For the transport, these are quite a bit more complex than the initialization UDCs. See Session 10 of the `ESP` training[8] for complete details.

Component UDCs are added to the system by updating `transport_init.udc`, as shown in Figure 11.

UDCs for each View

These UDCs are responsible for building or restoring the various needed Bodys, adding view-specific analysis attributes, providing a `_name`, and leaving all associated Bodys on the stack. An examples of this type of UDC is given in Figures 12. View UDCs are incorporated by adding appropriate lines to `transport.csm`, such as shown in Figure 13.

Building Up Configuration in Multiple Versions

Even with the above organization scheme, writing a very large model can be daunting. This is compounded by the fact that one does not always know beforehand the details of the design. Therefore it has been found that building up the configuration incrementally has been very effective.

For the transport configuration, the various versions that were built are:

1. add `transport.csm`, `transport_init.udc`, `wingPmtrs.udc`, `wingCalc.udc`, `wingOml.udc*`, and `viewConcept.udc*`
2. add `viewVlm.udc*`
3. add `wingHinges.udc*`

4. add `viewCfdInviscid.udc*`
5. add `wingWaffle.udc*` and `wingBem.udc*`, `viewBem.udc*`
6. add `htailPmtrs.udc`, `htailCalc.udc`, `htailOml.udc*`, `htailHinges.udc*`, and `htailVlm.udc*`
7. add `htailWaffle.udc*`, `htailBem.udc*`
8. add `vtailPmtrs.udc`, `vtailCalc.udc`, `vtailOml.udc*`, `vtailHinges.udc*`, and `vtailVlm.udc*`
9. add `vtailWaffle.udc*`, `vtailBem.udc*`
10. add `fusePmtrs.udc`, `fuseCalc.udc`, and `fuseOml.udc*`
11. add `fuseIml.udc*`, `fuseWaffle.udc*`, and `fuseBem.udc*`
12. add `nacellePmtrs.udc`, `nacelleCalc.udc`, `nacelleOml.udc*`, `pylonPmtrs.udc`, `pylonCalc.udc`, and `pylonOml.udc*`
13. add `payloadPmtrs.udc` and `payload.udc*`
14. add `viewCfdViscous.udc*`
15. add `CAPS` Attributes to all `view*` files
16. add `viewCantilever.udc*`, `viewSimplySupport.udc*`, and `viewSkins.udc*`

In the above discussion, the figures were created from version 5 (wing only). The associated models are shown in Figures 14 through 18.

Exploring a Model

With the generation of so many inter-connected models of so many components, it can be challenging to understand a model. This can either happen when encountering the model for the first time or when re-visiting a model that was created in the past. As a result, several new tracing tools have been recently added to `ESP`.

The first new way of tracing is to view the hierarchy of the files used in the model; Figure 19 is an example of such a call tree. The figure clearly shows that some UDCs are called several times from different places.

The second new way of tracing is to look at the Bodys that are stored; This trace is generated by starting from a user-defined pattern. An example of the wing-related Bodys built for the "concept" view of the transport is shown in Figure 20.

The third new way of tracing is to focus on the various parameters that are shared by the various models (in the various UDCs). Figure 21 shows the result of tracing all the wing-related parameters for the transport.

Finally, a user can trace through the attributes of a configuration, such as shown in Figure 22, which shows the places where the `gap` attributes are defined.

Experience has shown that these new tracing tools can greatly facilitate the understanding of a complex model such as the transport.

Case Study 2: Canard Fighter

Modern complex geometry representations require these entities to be structured for multiple-uses, collaborative-friendly, and be able to update as the analysis and design process evolves. Thus, setting-up the script based on specific guidelines is crucial to facilitate this ever changing and evolving need. To explore the proposed management scheme in the effort, a legacy "dusty-deck" model was converted into this new scheme.

The "dusty-deck" representation was scripted during the first phase of the CAPS project and the outcomes were published in[9] and.[10] As a summary, firstly, the legacy script was a single giant file with close to two thousand lines of code. Secondly, at this phase, the interface to allow automated linking of geometry to meshing and analysis was still in the development stage. So, the representation did not have the notion of "analysis views", described above. And thirdly, the build time to generate a water-tight representation was around 10 minutes. The script is still part of the ESP test suite, to check that the legacy features still work, as backward compatibility is one of the philosophical features of the ESP development. So, the goal of this exercise was to incorporate the enhancements, features as well as conveniences to modernize the script status and enable future improvements with ease.

At present, all the geometry features have been transferred into the new scheme. The management scheme used for the transport configuration, described in the previous sections, was used as a template. This greatly accelerated the process of conversion. This also allowed to re-use the attribution scheme as much as possible, so the UDCs that generate analysis views required minimal modifications. Finally, the template also enabled incorporating all the new scripting language features and geometry enhancements, as the examples were readily available in the template and easily convertible from transport configuration to update this suite of legacy scripts.

The current state of the organized script-suite is illustrated here in the Figures 23 through 25 with three views: (a) Concept View: shows that the transfer of all the geometry features from the dusty-deck to the modern scheme, (b) a complete VLM View: which was partially implemented in the original script, and (c) a CFD View: that incorporates the improved attribution and geometry features that enable automated mesh generation and CFD analysis (suitable for SU2, FUN3D or Cart3D[11] solvers, as an example) workflow. This updated representation gets build in around 2 minutes (compared to 10 minutes with old version of the script, using the old version of ESP). This demonstrates the enhancements that have been implemented in ESP, including updated version of OpenCASCADE. In summary, realizing the benefits of using the proposed management scheme, the on-going effort is to convert all the previous scripts into this new scheme. This will enable a consistent user experience, making these scripts easy to understand and update by script developers as well as ESP users.

Summary

A new method for managing geometric models for multi-component, multi-disciplinary analysis and design is described. It is based upon the use of user-defined components (UDCs) that are organized to create the various sub-models that are needed. All of this in enabled by `ESP`, which supports multiple linked models, persistent attribution, and sensitivities.

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]Neill D.J., Johnson, E.H., Canfield, R., "ASTROS - A multidisciplinary automated structural design tool", Journal of Aircraft, Vol. 27, No. 12, Decemeber 1990.

[4]"NASTRAN Analysis Quick Reference Manual," MSC Software, Santa Ana, CA 92707

[5]Economon, T. D., Palacios, F., Copeland, S.R., Lukaczyk, T.W., and Alonso J.J., "SU2: An Open-Source Suite for Multiphysics Simulation and Design", AIAA Journal Vol. 54, No. 3, March 2016.

[6]Anderson, W.K., Biedron, R.T., Carlson, J-R, Derlaga, J.M., Druyor Jr. C.T., Gnoffo, P.A., Hammond, D.P., Jacobson, K.E., Jones, W.T., Kleb, B., Lee-Rausch, E.M., Nastac, G.C., Nielsen, E.J., Park, M.A., Rumsey, C.L., Thomas, J.L., Thomp-

son, K.B., Walden, A.C., Wang, L., Wood, S.L., Wood, W.A., Diskin, B., Liu, Y., Zhang, X., "FUN3D Manual: 14.0", NASA TM 20220017743, 2022.

[7]Drela, M., and Youngren, H., "AVL",, 2017. URL http://web.mit.edu/drela/Public/web/avl/.

[8]ESP training, cdl.mit.edu/ESP/training, December2022.

[9]Bhagat, N.D., Allison, D., Alyanak, E.J., "Geometry Driven High Fidelity Stability Derivatives Using An Automated CFD Analysis Process", AIAA 2015-2648, June 2015.

[10]Bhagat, N., Alyanak E., "Computational Geometry for Multi-fidelity and Multi-disciplinary Analysis and Optimization", AIAA-2014-0188, January, 2014.

[11]Aftosmis, M. J., Berger, M.J., Nemec, "Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry", AIAA Journal, 36(6):952-960, June 1998.

Table of contents:

	bem	cfdInviscid	cfdViscous	concept	vlm
fuse	X	X	X	X	
htail	X	X	X	X	X
nacelle		X	X	X	
payload				X	
pylon		X	X	X	
vtail	X	X	X	X	X
wing	X	X	X	X	X

Figure 1. Table of contents for the transport configuration

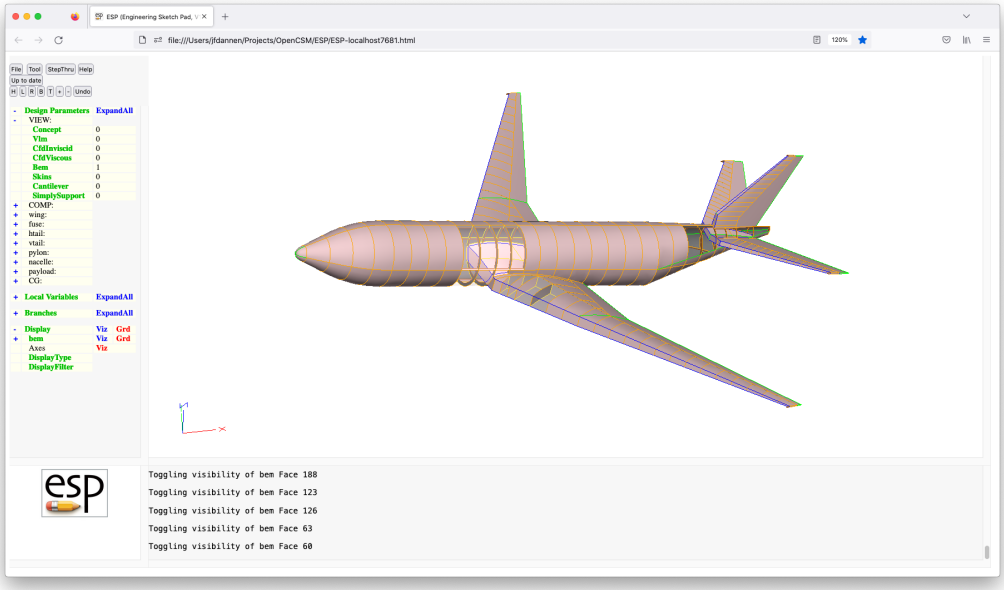

Figure 2. "bem" view of transport, with some transparent panels

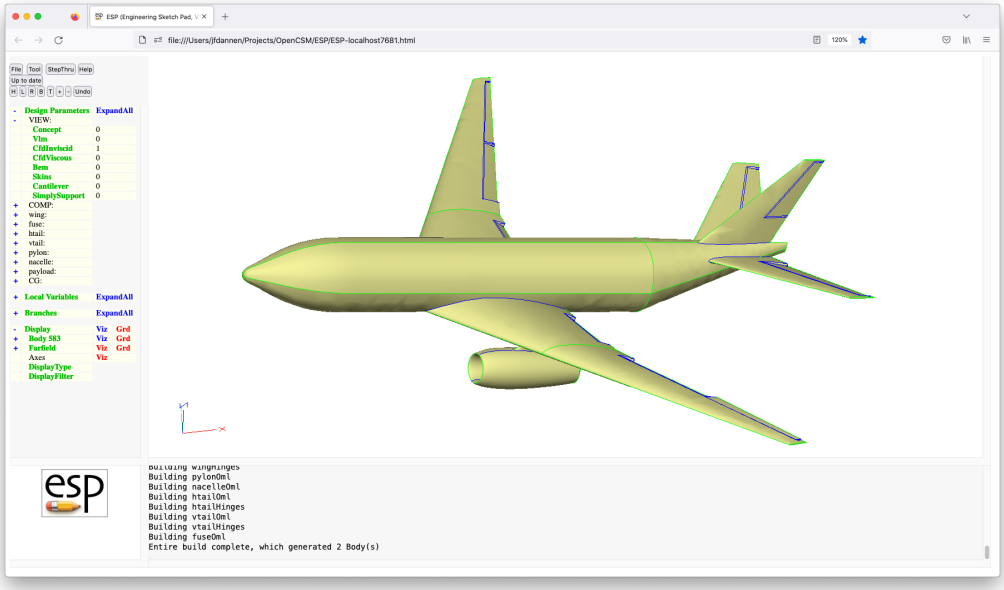

Figure 3. "cfdInviscid" view of transport, with farfield box removed

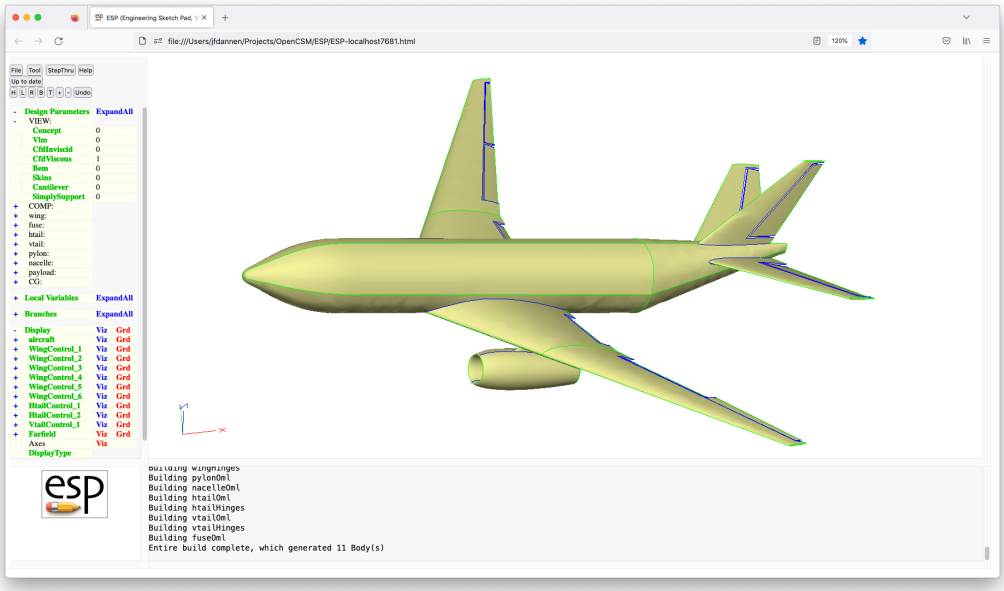

Figure 4. "cfdViscous"view of transport, with farfield box removed

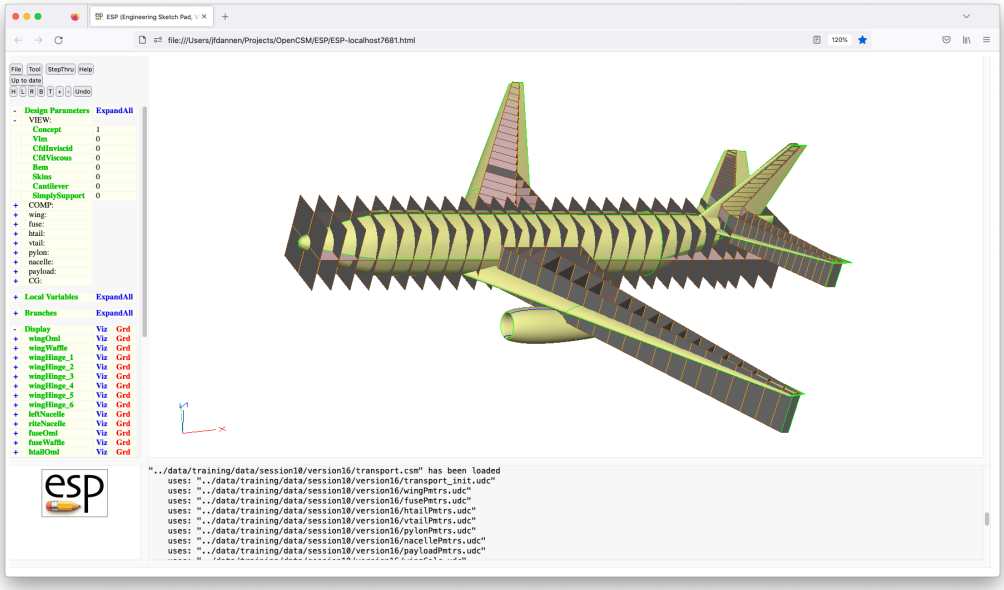

Figure 5. "Concept" view of transport, showing all primitives

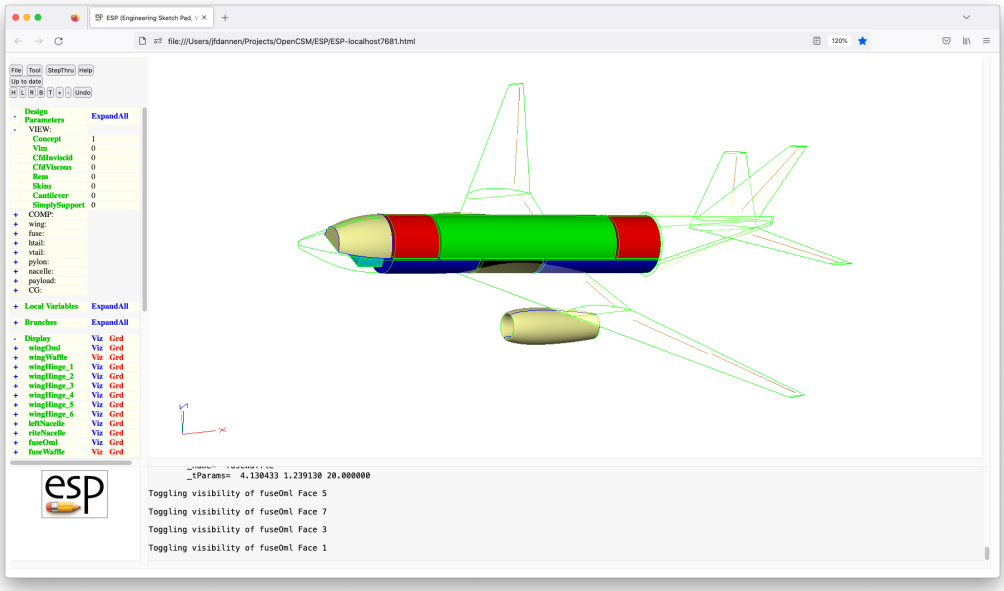

Figure 6. "Concept" view of transport, with waffle off and transparent OML

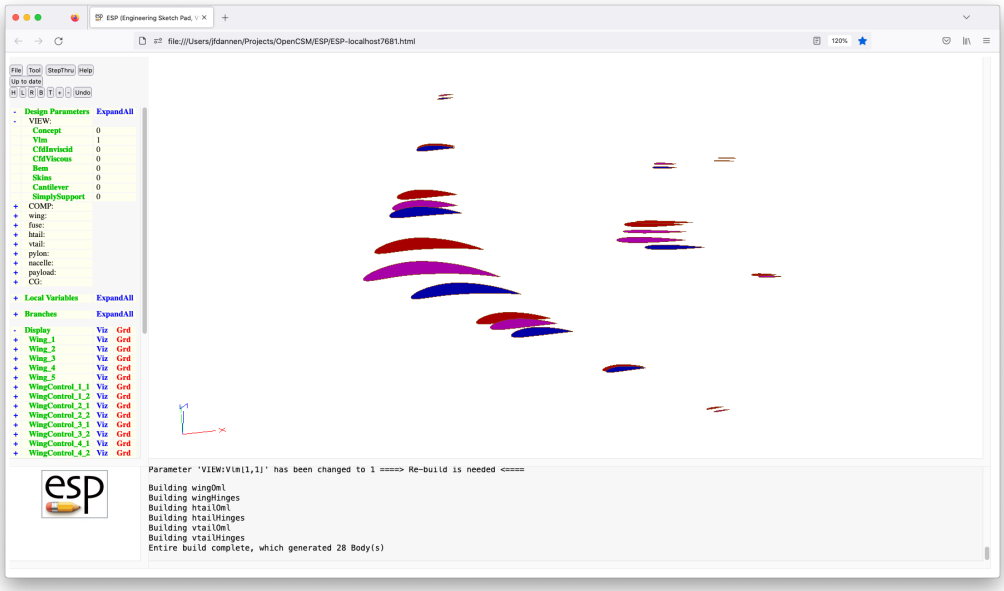

Figure 7. "vlm" view of transport

```
# .udc to make the Concept view
# written by John Dannenhoffer

INTERFACE . ALL

# make sure we have the necessary Bodys
IFTHEN    COMP:wing NE 0
    UDPRIM    $/wingOml
    UDPRIM    $/wingWaffle
    UDPRIM    $/wingHinges
ENDIF

# now that we have all the Bodys, show them
IFTHEN    COMP:wing NE 0
   RESTORE    wingOml
      ATTRIBUTE _name $wingOml
   RESTORE    wingWaffle
      ATTRIBUTE _name $wingWaffle
   PATBEG     ihinge  wing:hinge.nrow*COMP:controls
      RESTORE wingHinge ihinge
   PATEND
ENDIF

END
```

Figure 8. Listing of `viewConcept.udc`

```
# .udc to define the DESPMTRs and CFGPMTRs for a wing
# written by John Dannenhoffer

INTERFACE . ALL

# wing Oml
DESPMTR   wing:area        4240    # area
DESPMTR   wing:aspect      9.00    # aspect ratio
DESPMTR   wing:taperi      0.48    # inboard  taper ratio
DESPMTR   wing:tapero      0.23    # outboard taper ratio
DESPMTR   wing:sweep       35.0    # leading edge sweep
DESPMTR   wing:dihedral     7.0    # dihedral
DESPMTR   wing:break       0.37    # inboard/outboard
DESPMTR   wing:alphar      -1.0    # setting angle   at root
DESPMTR   wing:thickr      0.10    # thickness ratio at root
DESPMTR   wing:camberr     0.08    # camber    ratio at root
DESPMTR   wing:alphab      -3.0    # setting angle   at break
DESPMTR   wing:thickb      0.15    # thickness ratio at break
DESPMTR   wing:camberb     0.04    # camber    ratio at break
DESPMTR   wing:alphat      -8.0    # setting angle   at tip
DESPMTR   wing:thickt      0.08    # thickness ratio at tip
DESPMTR   wing:cambert     0.01    # camber    ratio at tip
DESPMTR   wing:xroot       50.0    # xloc at root LE
DESPMTR   wing:zroot       -8.0    # zloc at root LE

# wing hinge lines
DIMENSION wing:hinge        6 9 1 # ymin                  ymax
#                           theta   x/c   y/span  z/t     x/c   y/span z/t    gap   grp
DESPMTR   wing:hinge       "-10.0;  0.75; -0.98;  0.50;   0.75; -0.70; 0.50;  0.25; 1;  \ left aileron
                            +10.0;  0.75; -0.69;  0.00;   0.75; -0.43; 0.00;  0.25; 2;  \ left oflap
                            +15.0;  0.85; -0.33;  0.00;   0.90; -0.14; 0.00;  0.25; 3;  \ left iflap
                            +15.0;  0.90;  0.14;  0.00;   0.85;  0.33; 0.00;  0.25; 3;  \ rite iflap
                            +10.0;  0.75;  0.43;  0.00;   0.75;  0.69; 0.00;  0.25; 2;  \ rite oflap
                            +10.0;  0.75;  0.70;  0.50;   0.75;  0.98; 0.50;  0.25; 4"  # rite aileron

# wing structure
DESPMTR   wing:spar1       0.20    # fraction of chord for LE spar
DESPMTR   wing:spar2       0.70    # fraction of chord for TE spar
CFGPMTR   wing:nrib1          2    # number of internal ribs in region 1
CFGPMTR   wing:nrib2          4    # number of internal ribs in region 1
CFGPMTR   wing:nrib3         12    # number of internal ribs in region 1

DESPMTR   wing:waffleGap      1    # distance between fuselage and wing root rib

DESPMTR   wing:dxnom        2.0    # nominal .bdf element side length

END
```

Figure 9. Listing of `wingPmtrs.udc`

```
# .udc to calculate critial locations and dimensions for a wing
# written by John Dannenhoffer

INTERFACE . ALL

OUTPMTR   wing:mac
OUTPMTR   wing:span

SET       wing:span      sqrt(wing:aspect*wing:area)
SET       wing:yroot     0
SET       wing:ytip     -wing:span/2
SET       wing:xtip      wing:xroot-wing:ytip*tand(wing:sweep)
SET       wing:ztip      wing:zroot-wing:ytip*tand(wing:dihedral)
SET       wing:ybreak    wing:ytip*wing:break
SET       wing:xbreak    wing:xroot-wing:ybreak*tand(wing:sweep)
SET       wing:zbreak    wing:zroot-wing:ybreak*tand(wing:dihedral)
SET       wing:chordr    wing:area/((wing:yroot-wing:ybreak)*(wing:taperi+1)\
                         +(wing:ybreak-wing:ytip)*wing:taperi*(wing:tapero+1))
SET       wing:chordb    wing:chordr*wing:taperi
SET       wing:chordt    wing:chordb*wing:tapero
SET       wing:mac       sqrt(wing:area/wing:aspect)
SET       wing:sharpte   SHARP_TE

END
```

Figure 10. Listing of `wingcalcs.udc`

```
# .udc to set up DESPMTRs, CFGPMTRs, and critical locations and dimensions
# written by John Dannenhoffer

INTERFACE . ALL

# global tolerance
set EPS06 1.0e-6

# make a list of the components
CFGPMTR   COMP:wing     1

# controls must be either 0=off or 1=on
CFGPMTR   COMP:controls 1
IFTHEN    COMP:controls NE 0  AND  COMP:controls NE 1
   MESSAGE  COMP:controls_must_be_0_or_1
   THROW    -999
ENDIF

# define the DESPMTRs and CFGPMTRs
UDPRIM    $/wingPmtrs

# put sharp trailing edges on all aero surfaces
SET       SHARP_TE   1

# compute critical locations / dimensions
UDPRIM    $/wingCalc

# CG location used to drive design parametres, not the actual CG
DIMENSION CG:ref  3 1
DESPMTR   CG:ref  "90; 0; 0"

END
```

Figure 11. Listing of `transport_init.udc`

```
# .udc to make the Bem
# written by John Dannenhoffer

INTERFACE . ALL

# make sure we have the wingBem
IFTHEN    COMP:wing   NE 0
   UDPRIM    $/wingBem
ENDIF

IFTHEN    COMP:wing   NE 0
   RESTORE   wingBem
      ATTRIBUTE _name $wingBem
ENDIF

END
```

Figure 12. Listing of `viewBem.udc`

```
# transport
# written by John Dannenhoffer

# define the views
CFGPMTR   VIEW:Concept          1
CFGPMTR   VIEW:Vlm              0

CFGPMTR   VIEW:CfdInviscid      0

CFGPMTR   VIEW:Bem              0

UDPRIM    $/transport_init

IFTHEN    VIEW:Concept      NE  0
   UDPRIM    $/viewConcept
ENDIF

IFTHEN    VIEW:Vlm          NE  0
   UDPRIM    $/viewVlm
ENDIF

IFTHEN    VIEW:CfdInviscid  NE  0
   UDPRIM    $/viewCfdInviscid
ENDIF

IFTHEN    VIEW:Bem          NE  0
   UDPRIM    $/viewBem
ENDIF

END
```

Figure 13. Listing of `transport.csm`

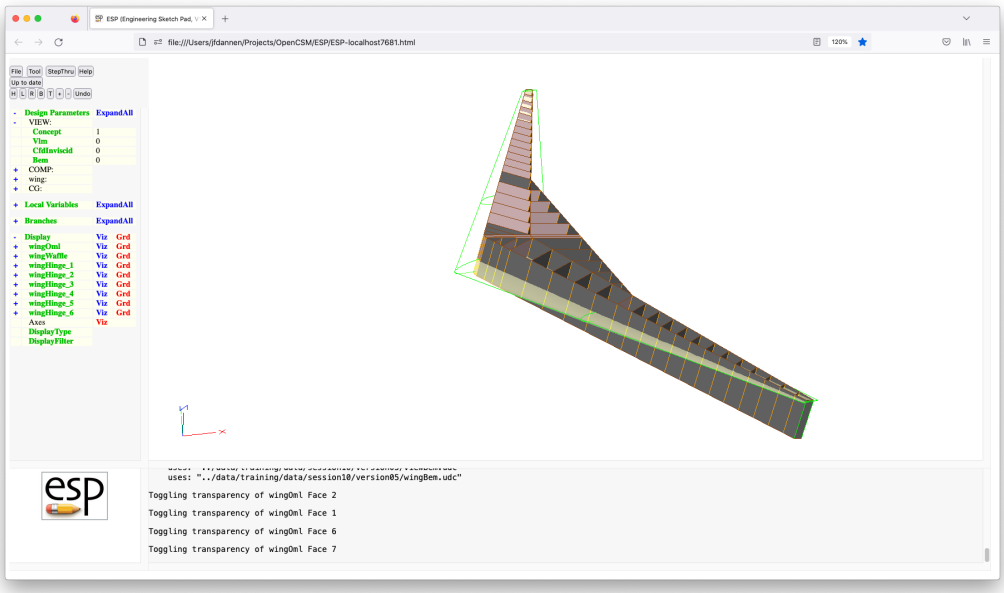

Figure 14. Concept view for version 5 of the transport (wing only), showing all primitives

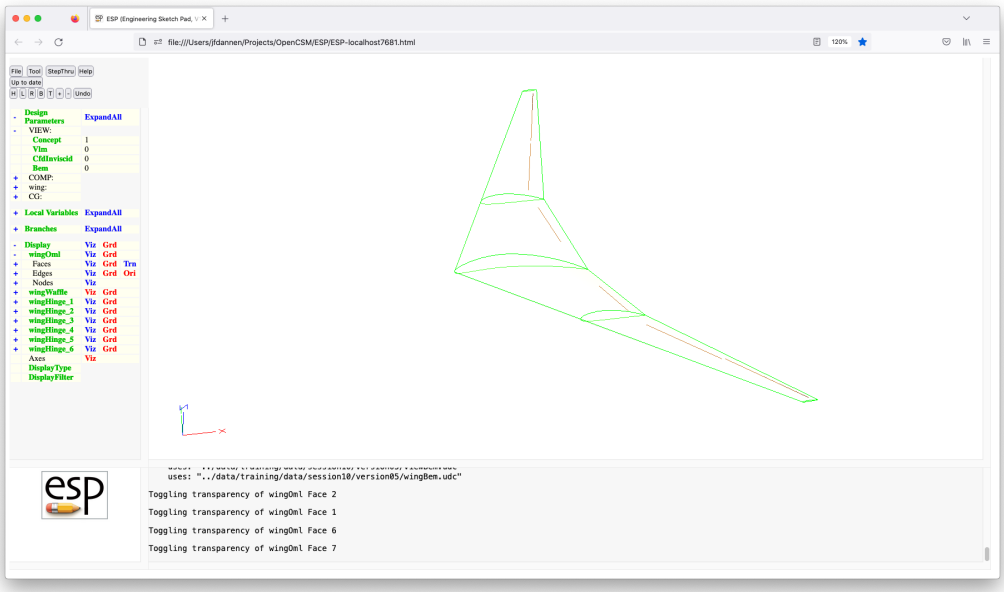

Figure 15. Concept view for version 5 of the transport (wing only), with waffle off and transparent OML

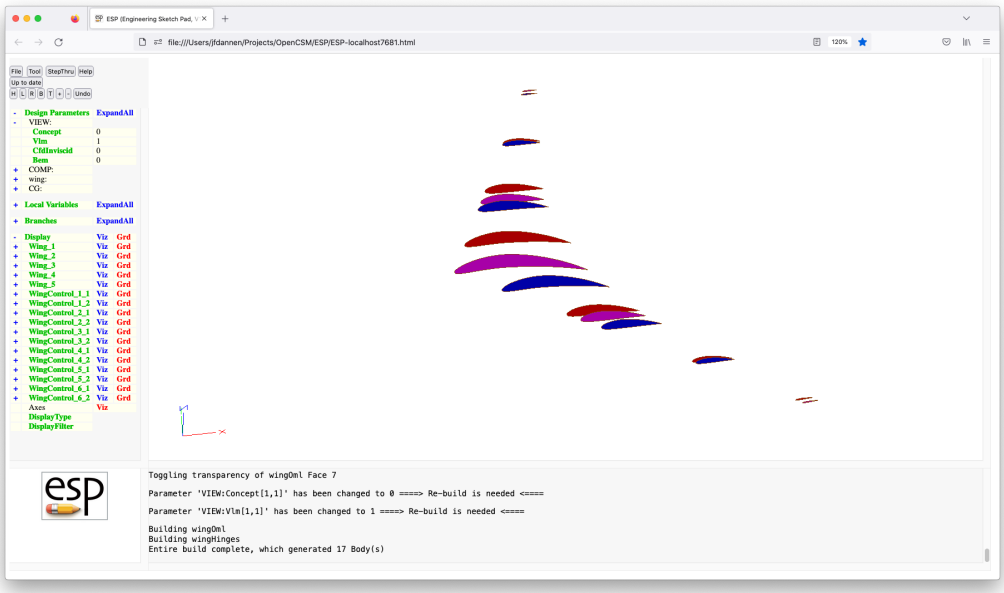

Figure 16. Vlm view for version 5 of the transport (wing only)

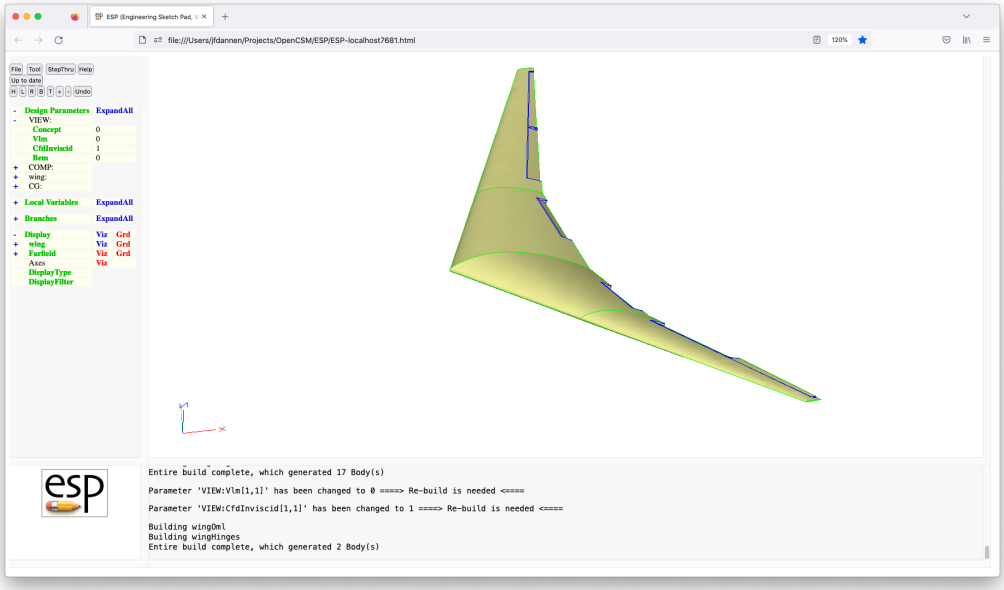

Figure 17. CfdInviscid view for version 5 of the transport (wing only)

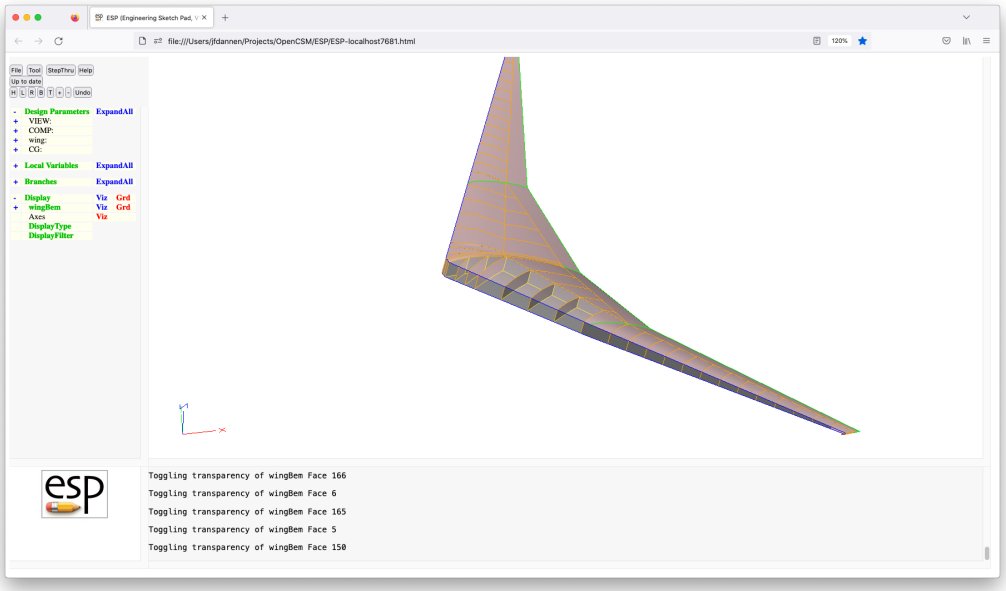

Figure 18. Bem view for version 05 of the transport (wing only), with some transparent panels

```
Files used in previous build:
[[../data/BigModels/transport/version14/transport.csm:1]]
    [[../data/BigModels/transport/version14/transport_init.udc:1]]
        [[../data/BigModels/transport/version14/wingPmtrs.udc:1]]
        [[../data/BigModels/transport/version14/fusePmtrs.udc:1]]
        [[../data/BigModels/transport/version14/htailPmtrs.udc:1]]
        [[../data/BigModels/transport/version14/vtailPmtrs.udc:1]]
        [[../data/BigModels/transport/version14/pylonPmtrs.udc:1]]
        [[../data/BigModels/transport/version14/nacellePmtrs.udc:1]]
        [[../data/BigModels/transport/version14/payloadPmtrs.udc:1]]
        [[../data/BigModels/transport/version14/wingCalc.udc:1]]
        [[../data/BigModels/transport/version14/htailCalc.udc:1]]
        [[../data/BigModels/transport/version14/vtailCalc.udc:1]]
        [[../data/BigModels/transport/version14/fuseCalc.udc:1]]
        [[../data/BigModels/transport/version14/pylonCalc.udc:1]]
            [[../data/BigModels/transport/version14/nacelleCalc.udc:1]]
    [[../data/BigModels/transport/version14/viewConcept.udc:1]]
        [[../data/BigModels/transport/version14/wingOml.udc:1]]
        [[../data/BigModels/transport/version14/wingWaffle.udc:1]]
            [[../data/BigModels/transport/version14/wingOml.udc:1]]
        [[../data/BigModels/transport/version14/wingHinges.udc:1]]
            [[../data/BigModels/transport/version14/wingOml.udc:1]]
        [[../data/BigModels/transport/version14/nacelleOml.udc:1]]
        [[../data/BigModels/transport/version14/htailOml.udc:1]]
        [[../data/BigModels/transport/version14/htailWaffle.udc:1]]
            [[../data/BigModels/transport/version14/htailOml.udc:1]]
        [[../data/BigModels/transport/version14/htailHinges.udc:1]]
            [[../data/BigModels/transport/version14/htailOml.udc:1]]
        [[../data/BigModels/transport/version14/vtailOml.udc:1]]
        [[../data/BigModels/transport/version14/vtailWaffle.udc:1]]
            [[../data/BigModels/transport/version14/vtailOml.udc:1]]
            [[../data/BigModels/transport/version14/htailWaffle.udc:1]]
        [[../data/BigModels/transport/version14/vtailHinges.udc:1]]
            [[../data/BigModels/transport/version14/vtailOml.udc:1]]
        [[../data/BigModels/transport/version14/fuseOml.udc:1]]
        [[../data/BigModels/transport/version14/fuseWaffle.udc:1]]
            [[../data/BigModels/transport/version14/fuseOml.udc:1]]
            [[../data/BigModels/transport/version14/fuseIml.udc:1]]
            [[../data/BigModels/transport/version14/wingWaffle.udc:1]]
            [[../data/BigModels/transport/version14/htailWaffle.udc:1]]
        [[../data/BigModels/transport/version14/payload.udc:1]]
            [[../data/BigModels/transport/version14/fuseIml.udc:1]]
```

Figure 19. Call-tree for transport

```
Trace of Storages matching "wing*":
    wingOml  0
        made in [[../data/BigModels/transport/version14/wingOml.udc:95]]
        used in [[../data/BigModels/transport/version14/wingOml.udc:14]]
        used in [[../data/BigModels/transport/version14/wingOml.udc:14]]
        used in [[../data/BigModels/transport/version14/wingWaffle.udc:34]]
        used in [[../data/BigModels/transport/version14/wingOml.udc:14]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:29]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:34]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:41]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:46]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:40]]
    wingWaffle  0
        made in [[../data/BigModels/transport/version14/wingWaffle.udc:153]]
        used in [[../data/BigModels/transport/version14/wingWaffle.udc:12]]
        used in [[../data/BigModels/transport/version14/wingWaffle.udc:12]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:42]]
    wingHinge  1
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
    wingHinge  2
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
    wingHinge  3
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
    wingHinge  4
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
    wingHinge  5
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
    wingHinge  6
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
```

Figure 20. Trace of wing-related Bodys created during the build of the "concept" view of the transport

```
Trace of Storages matching "wing*":
    wingOml  0
        made in [[../data/BigModels/transport/version14/wingOml.udc:95]]
        used in [[../data/BigModels/transport/version14/wingOml.udc:14]]
        used in [[../data/BigModels/transport/version14/wingOml.udc:14]]
        used in [[../data/BigModels/transport/version14/wingWaffle.udc:34]]
        used in [[../data/BigModels/transport/version14/wingOml.udc:14]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:29]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:34]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:41]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:46]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:40]]
    wingWaffle  0
        made in [[../data/BigModels/transport/version14/wingWaffle.udc:153]]
        used in [[../data/BigModels/transport/version14/wingWaffle.udc:12]]
        used in [[../data/BigModels/transport/version14/wingWaffle.udc:12]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:42]]
    wingHinge  1
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
    wingHinge  2
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
    wingHinge  3
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
    wingHinge  4
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
    wingHinge  5
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
    wingHinge  6
        made in [[../data/BigModels/transport/version14/wingHinges.udc:53]]
        used in [[../data/BigModels/transport/version14/wingHinges.udc:17]]
        used in [[../data/BigModels/transport/version14/viewConcept.udc:45]]
```

Figure 21. Trace of wing-related parameters associated with the transport

```
Trace of Attributes matching "gap":
    "gap" applied to [[../data/BigModels/transport/version14/wingHinges.udc:54]]
    "gap" applied to [[../data/BigModels/transport/version14/htailHinges.udc:54]]
    "gap" applied to [[../data/BigModels/transport/version14/vtailHinges.udc:54]]
```

Figure 22. Trace of "gap" attribute for the transport

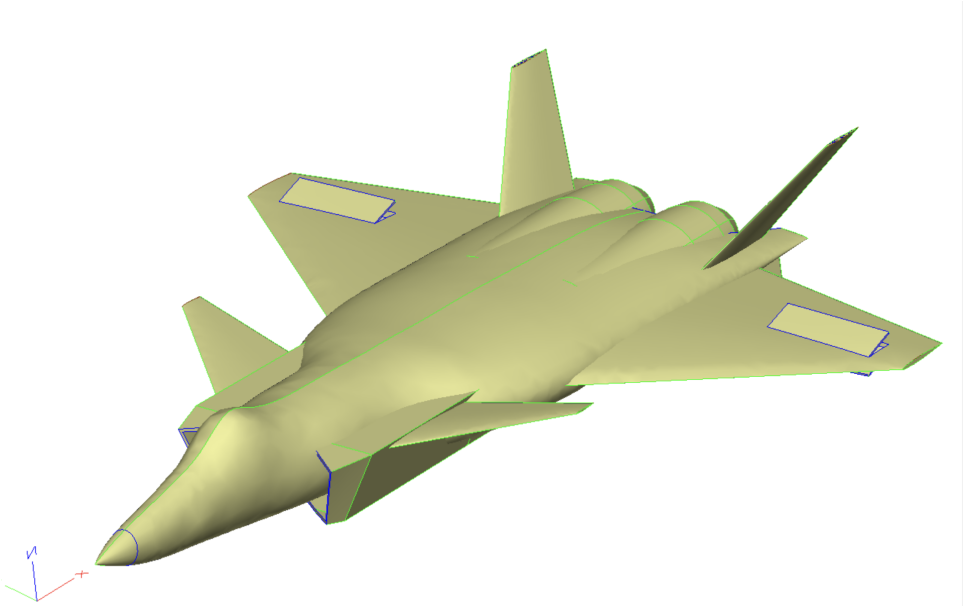

Figure 23. Concept view for canard fighter

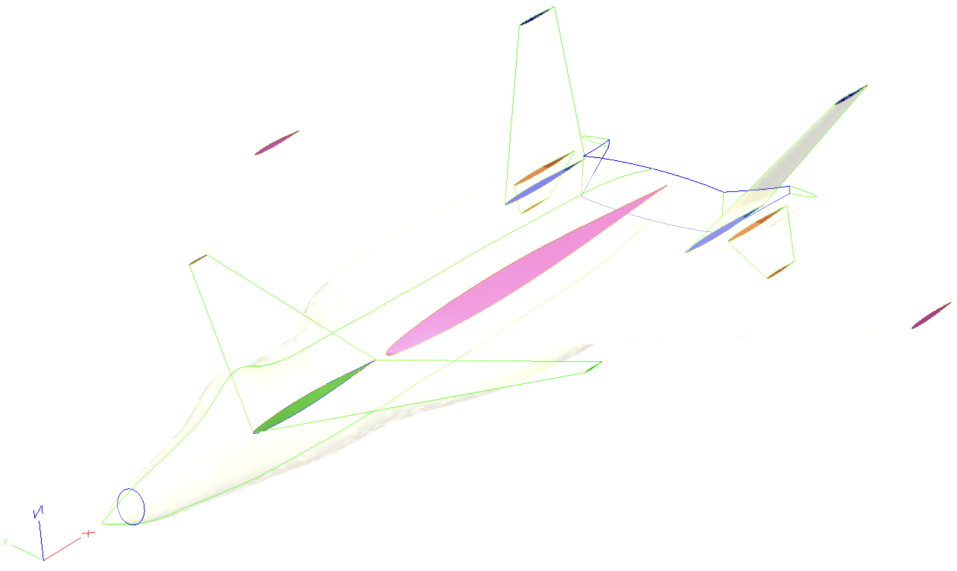

Figure 24. Vlm view for canard fighter

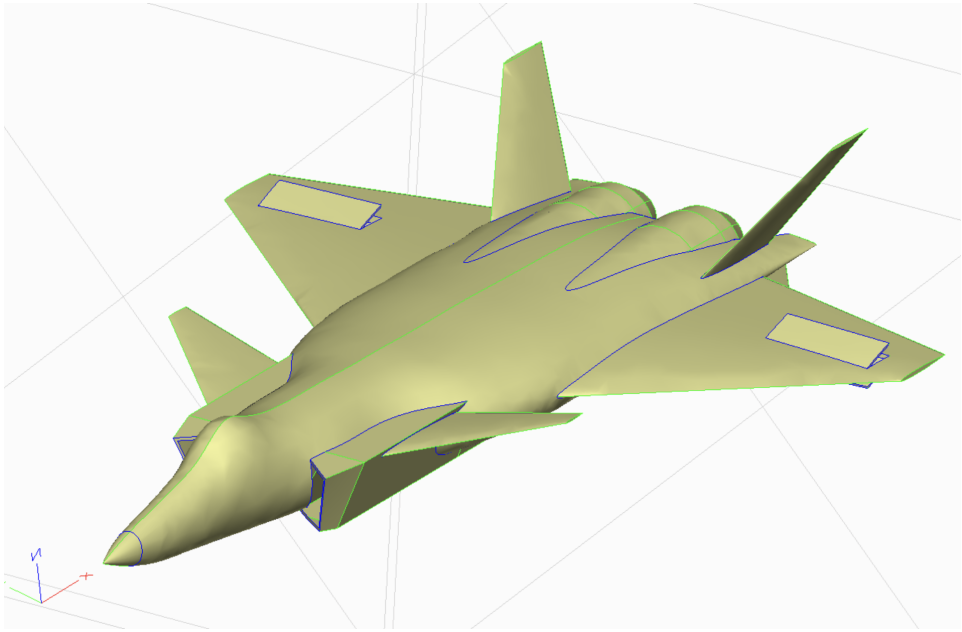

Figure 25. CFD view for canard fighter