Generation of Multi-fidelity, Multi-discipline Air Vehicle Models with 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

Efficient multi-disciplinary analysis and optimization (MDAO) of aerospace vehicles is enabled by the generation of parametric, feature-based models. Many systems for generating these vehicles have been developed over the past decades, but most of these systems focus on the development of models that are well suited to the manufacturing process. Recently, the Engineering Sketch Pad (`ESP`) was introduced whose expressed purpose is the generation of models for aerospace design and analysis. Three of the key features of `ESP` are its ability to develop multiple, linked models for use throughout the design process, to utilize user-defined features that are central to aerospace vehicles (such as airfoils, flaps, and spoilers), and to compute analytic sensitivities for optimization and uncertainty quantification.

Contained herein is a description of `ESP`, including those components that directly support the above. In particular, the use of `ESP` to generate models within a single discipline that are used in the conceptual, preliminary, and final design stages is reviewed. Similarly, multiple linked models in cooperating disciplines are also discussed. These concepts are demonstrated on a sample transport configuration.

I. Background

Since the early 1960s,[1] there have been an increasingly-complex (complicated) series of "CAD" systems to support the geometry needs of the manufacturers of mechanical devices. In the early years, "CAD" was an acronym for "Computer-Aided Drafting". These early "CAD" systems were built to allow a drafter to draw with pointing device (dial boxes, trackballs, light pens and etc.) onto a computer screen in the same way that the drafter had traditionally drawn with a pencil on a drafting table. The key advantage of the "CAD" systems were that it was easy to make lines straight, circles round, changes could be easily made, and errors could be erased. The early "CAD" systems were two dimensional (just like the drafting board), and knew nothing about the third dimension. The drafter or person reading the drawing needed to convert the two-dimensional representation into a three-dimensional concept.

In the next generation of "CAD" system, the acronym was changed to "Computer-Aided Drawing". Here systems evolved from two dimensions to three dimensions, thereby giving the drafter the ability to create multiple orthographic projections of the same object. For this additional capability, the drafter was required

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

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

to prescribe three-dimensional coordinates for everything that was drawn. Typically this was performed by the establishment of a sketching plane, on which the drawing was done. These new "Drawing" systems had a great advantage over the "Drafting" systems, but still only created wireframes. Because the "Drawing" systems did not understand the concepts of a solid, it was possible to create nonsensical wireframes that could not be built.

The acronym "CAD" once again changed into "Computer-Aided Design" in the next generation of systems, in which the concept of a *solid* first emerged. This was done by taking the perspective that one starts with a (possibly simple) solid body, and then performs operations on that body until it achieved its ultimate final shape. The types of operations that could be performed included adding and removing material (which was described in terms of its own solid body), and by morphing the basic shape. At about the same time, the concept of *parameters* was added to "CAD" systems, making it possible to create a model whose dimensions could be easily modified. The robustness of this latter capability depended strongly on the types of operations that the model builder (formerly drafter) used in constructing the model.

Since the "Computer-Aided Design" systems took the perspective of adding, removing, or morphing some shape, users of the systems are encouraged to "make your models mimic the manufacturing process".[2] If this is done, the generation of manufacturing processes, such as tool-path creation, flat-panel plans, weldments, or mold tools, can be automated. Hence, it is probably best to classify these "CAD" systems as "mCAD" systems.

But for the (optimal) design of complex systems, the analytical designer wants to think about the function and performance of the device being generated, requiring the generation of a separate "aCAD" model. Sometimes the systems that produce analysis-ready models are thought of as "pre-preprocessing", referring to the order utilized in the larger design/manufacturing process ("aCAD" before "mCAD"). It is interesting to note that almost all "aCAD" systems are (and have always been) parametric but "mCAD" has only been parametric over the last two decades and most models currently built in "mCAD" systems are not parametrically driven.

Good design is driven by a suite of parameters that is both familiar to practitioners (in the various disciplines) and are flexible enough to allow for finding the set of parameter values that meet, or exceed, the stated mission. This allows for achieving the goal in design and facilitates learning about the decisions and choices made in an automated optimization setting.

A early and effective example of an "aCAD" system is Boeing's Aero-Grid and Paneling System (`AGPS`),[3] which was developed for the use with specific analysis codes. As stated above, the modeling techniques supported by "aCAD" and "mCAD" are rather dissimilar, and so transfer between them is done by limited translators or by starting over. This one-way path from "aCAD" to "mCAD" leads to a design process where real feedback is not really possible.

II. The Engineering Sketch Pad (ESP)

The "Engineering Sketch Pad" (`ESP`) system is a geometry creation and manipulation system, designed specifically to support the analysis and design of aerospace vehicles. It can be run stand-alone for the development of models, or it can be embedded into other analysis and design systems to support their geometry needs. It can also communicate with "mCAD" system through the use of STEP files to transmit the *finished* geometry from a design study.

Unlike modern "mCAD" systems (such as `UniGraphics`, `Catia`, `Pro/ENGINEER`, `SolidWorks`, or `AutoCAD`), `ESP` is not a full-featured system designed to support the mechanical design and manufacturing of complex parts. It is also not a system to be used for creating "drawings", nor can it generate manufacturing plans. `ESP` is currently being used at NASA, in the U.S. Air Force, and other organizations, especially to generate multi-fidelity and multi-disciplinary geometries to support analysis and design within current (and future) MDAO settings.

```
      # bolt example

      # design parameters
 1:   despmtr   Thead     1.00   # thickness of head
 2:   despmtr   Whead     3.00   # width     of head
 3:   despmtr   Fhead     0.50   # fraction  of head that is flat

 4:   despmtr   Dslot     0.75   # depth of slot
 5:   despmtr   Wslot     0.25   # width of slot

 6:   despmtr   Lshaft    4.00   # length   of shaft
 7:   despmtr   Dshaft    1.00   # diameter of shaft

 8:   despmtr   sfact     0.50   # overall scale factor

      # head
 9:   box       0      -Whead/2  -Whead/2  Thead   Whead     Whead
 9a:    attribute name $head
10:   rotatex   90  0  0
11:   box       0      -Whead/2  -Whead/2  Thead   Whead     Whead
11a:    attribute name $head
12:   rotatex   45  0  0
13:   intersect

14:   set       Rhead  (Whead^2/4+(1-Fhead)^2*Thead^2)/(2*Thead*(1-Fhead))
15:   sphere    0             0   0    Rhead
15a:    attribute name $head
16:   translate Thead-Rhead   0   0
17:   intersect

      # slot
18:   box       Thead-Dslot  -Wslot/2   -Whead   2*Thead   Wslot    2*Whead
18a:    attribute name $slot
19:   subtract

      # shaft
20:   cylinder  -Lshaft   0   0   0   0   0   Dshaft/2
20a:    attribute name $shaft
21:   union

22:   scale     sfact

23:   end
```

Figure 1. Sample file `bolt.csm`, operation indices (left side) are added for clarity.

A. ESP Example

The input to ESP is a .csm file that contains all the information needed to create a model or set of models. This is an ASCII file that is both human readable and parsed by ESP. To facilitate the discussion below, the example bolt.csm file is given in Fig. 1; the numbers at the left of each line are not actually part of the file, but have been added to aid in the discussions that follow. Fig 2 contains a picture of the Body that is produced when the bolt.csm file finished executing.

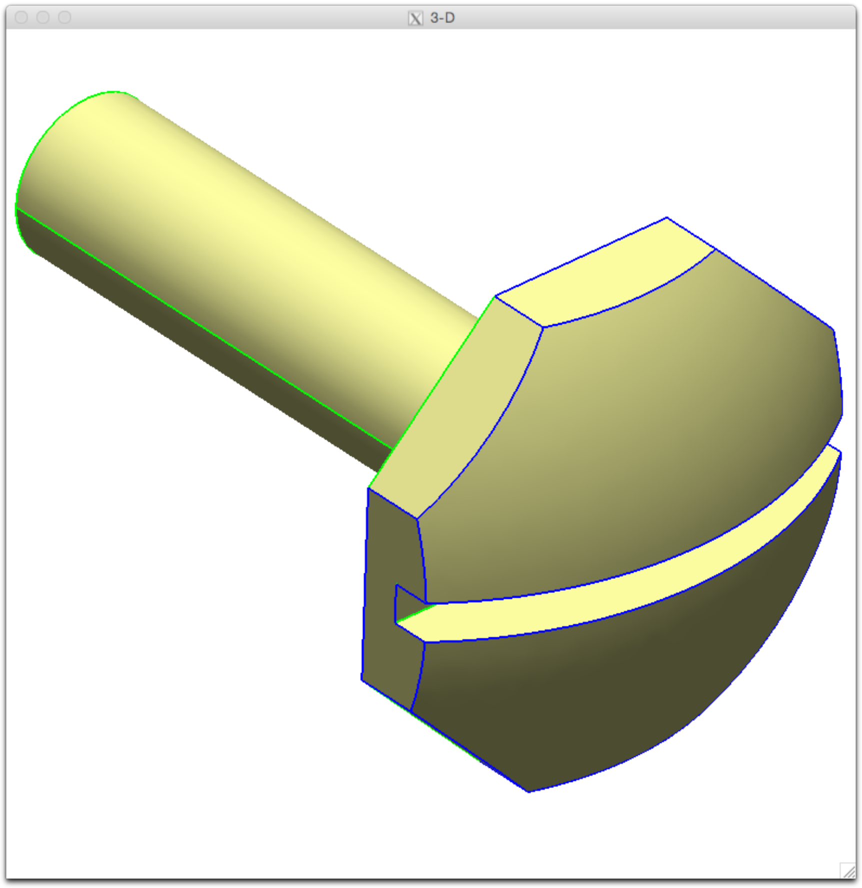


Figure 2. Body built by executing bolt.csm.

B. ESP's Distinguishing Features

In the next several paragraphs, the distinguishing features of ESP are described.

Solid Modeller

The ESP system[4] is built upon the OpenCSM constructive solid modeler,[5] which in built upon the "Engineering Geometry Aircraft Design System" (EGADS),[6] which in turn is built upon OpenCASCADE.[7] Because it uses a constructive solid modeling process, its models are guaranteed to be realizable solids (except as noted below). The fact that the solids are realizable makes the representation watertight, which is needed if one wants to create a computational grid for Computational Fluid Dynamics (CFD) or other 3D field analyses.

In certain models, ESP can generate non-manifold sheet Bodies (which can be used to model a mid-surface camber sheet, wake sheet, or thin structural plate) or wire Bodies (which can be used to model idealized wing spars or antennae).

The Master Models in `ESP` are defined in terms of a Feature Tree and a set of Design Parameters (see the *Parametric* section below).

The Feature Tree contains the recipe for constructing the configuration. It can be thought of as a (binary) tree that contains the sequence of operations needed to build a configuration. The Branches of the Feature Tree are defined by statements that specify the Branch type and a set of type-specific arguments (which may sometimes have default values). Branches are assigned default names when they are created, but the name can be be changed by the user to make the Feature Tree a bit more readable. If a Branch is attributed, its attribute(s) is/are assigned to all Edges and Faces that are created when the Branch executes. For-loops and if-blocks can be added to a Feature Tree through "patterns".

The Feature Tree that results from the `bolt.csm` file described above is shown in Fig. 3.

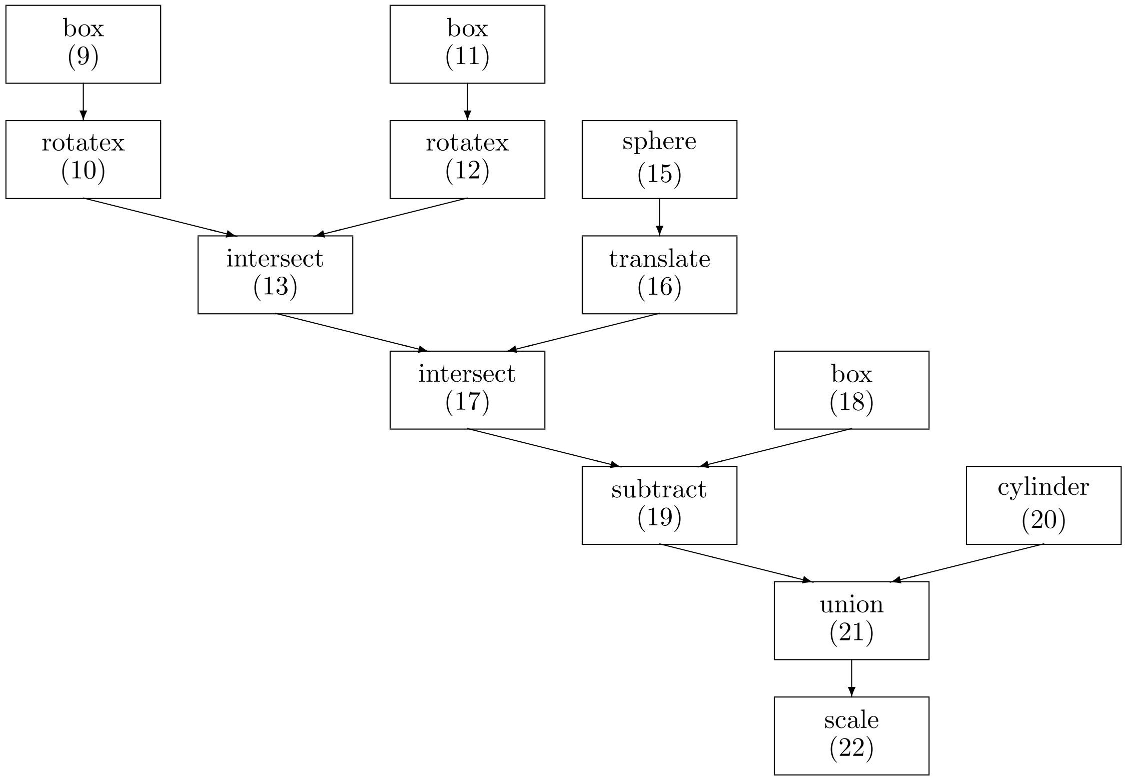


Figure 3. Feature Tree associated with `bolt.csm`.

Feature-based

All Master Models in `ESP` start with the generation of primitives. These include standard primitives (box, sphere, cone, cylinder, or torus), primitives grown from sketches (extrude, rule, blend, revolve, sweep, or loft), and user-defined primitives (UDPs). All of these primitive statements leave their resulting Body on a stack. In Fig. 1, operations at lines 9, 11, 15, 18, and 20 contain primitives and are designated as "0-1" in Table 1 (referring to consuming 0 stack entries and producing 1).

The Body on the top of the current stack can be be popped off the stack, transformed (translated, rotated, scaled, or mirrored), and the resulting Body is pushed back onto the stack. In Fig. 1, operations 10, 12, 16, 17, and 22 contain transformations which are labelled as "1-1" in Table 1.

There are other Feature Tree Branches that pop the Body from the top of the stack, apply a feature (such as fillet, chamfer, or hollow) to it, and then push the result back onto the stack. There are no applied features in Fig. 1 but they would also consume a single stack entry and produce one: "1-1".

It is also possible to combine the top two Bodies on the stack, via standard Boolean operations (intersect, subtract, or union). For example, to drill a hole in a Body, one creates a cylinder and then subtracts it from the Body on the top of the stack. Operations 13, 17, 19, and 21 contain Boolean operators in Fig. 1 and are designated as "2-1" in Table 1.

Table 1. `ESP` stack as filled during the operations seen in Fig. 1. Gray entries have no effect on the stack. The results of the operation are shown in boldface (at the top of the stack).

Operation	Usage	Stack bottom (#0)	Stack entry #1
1-8: despmtr	-		
9: box	0-1	**box**	
10: rotatex	1-1	90^o **rotated box**	
11: box	0-1	90^o rotated box	**box**
12: rotatex	1-1	90^o rotated box	45^o **rotated box**
13: intersect	2-1	**hexagonal slab**	
14: set	-	hexagonal slab	
15: sphere	0-1	hexagonal slab	**sphere**
16: translate	1-1	hexagonal slab	**translated sphere**
17: intersect	2-1	**hexagonal rounded head**	
18: box	0-1	hexagonal rounded head	**box**
19: subtract	2-1	**slotted head**	
20: cylinder	0-1	slotted head	**cylinder**
21: union	2-1	**bolt**	
22: scale	1-1	**scaled bolt**	

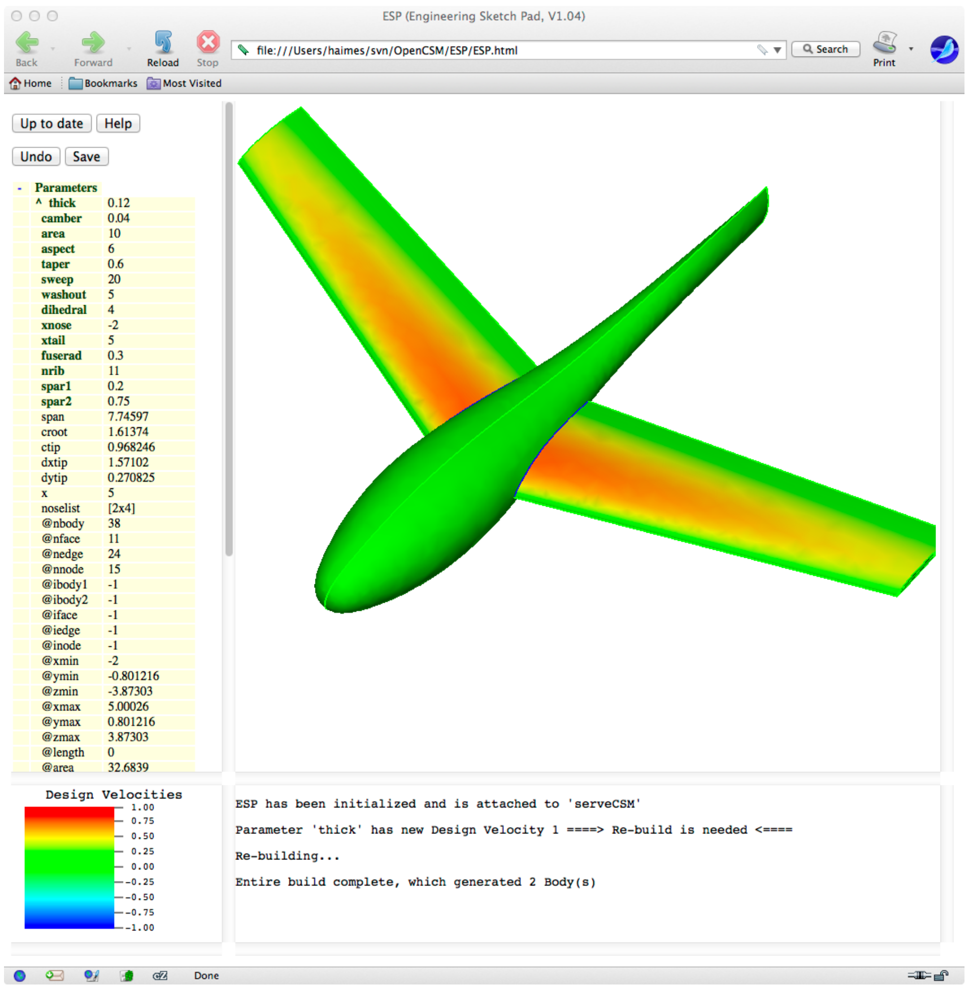
The stack usage during building the `bolt.csm` geometry can be seen in Table 1. It should be noted that the depth of the stack during the build is at most 2 entries for this simple construction.

Parametric

`ESP` Master Models typically contain one or more Design Parameters, which are single-valued, 1-D vectors, or 2-D arrays of numbers. Each Design Parameter has a current value, upper- and lower-bounds, and a current "velocity" (which is used to define sensitivities). Design Parameters can be "set" and retrieved ("get") either through `ESP`'s graphical user interface or externally via calls to it Application Programming Interface (API). In Fig. 1, the design Parameters are defined in lines 1 through 8.

`ESP`'s Design Parameters are used to set the arguments in the various Branches in the Feature Tree. In some cases the Design Parameters are used directly, and in others they are used in a MATLAB-like expression to set the argument. For example in line 9 in Fig. 1, one of the arguments is a number (`0`), three are simply the name of a Design Parameter (`Thead` twice and `Whead`), and two are expressions (`-Whead/2` twice).

An important function of an effective "aCAD" system is to provide the designer the ability to parameterize the model in ways appropriate for the mission "at hand". Some "aCAD" systems provide a collection of pre-parametrized components that can be easily assembled into a *concept*. This can quickly provide various concepts, but the results make it difficult to express the geometric range required for the design task. It is important for any general "aCAD" system to be able to allow the designer to easily express components using a parameterization that can be customized (this ability is at the *heart* of effective design). `ESP` allows for this by it's expression of the design in a simple scripting manner (as seen in Fig. 1) and by its ability to be *Extensible* in a parametric manner, as discussed below.



- Parameters	
^ thick	0.12
camber	0.04
area	10
aspect	6
taper	0.6
sweep	20
washout	5
dihedral	4
xnose	-2
xtail	5
fuserad	0.3
nrib	11
spar1	0.2
spar2	0.75
span	7.74597
croot	1.61374
ctip	0.968246
dxtip	1.57102
dytip	0.270825
x	5
noselist	[2x4]
@nbody	38
@nface	11
@nedge	24
@nnode	15
@ibody1	-1
@ibody2	-1
@iface	-1
@iedge	-1
@inode	-1
@xmin	-2
@ymin	-0.801216
@zmin	-3.87303
@xmax	5.00026
@ymax	0.801216
@zmax	3.87303
@length	0
@area	32.6839

```
ESP has been initialized and is attached to 'serveCSM'

Parameter 'thick' has new Design Velocity 1 ====> Re-build is needed <====

Re-building...

Entire build complete, which generated 2 Body(s)
```

Figure 4. Sensitivity of surface shape with respect to the wing's thickness for a glider configuration.

Differentiated

Probably the most distinctive feature of ESP is that it allows a user to compute the sensitivity of any part of a configuration with respect to any Design Parameter.[8] This is an essential capability when one wants to embed a geometry system into a gradient-based design optimization system.

For many of OpenCSM's commands, theses sensitivities are computed by analytically differentiating the Feature Tree. When this can be done, the process is extremely efficient, since there is no need to re-generate the configuration. In addition, the process produces sensitives the contain no truncation errors.

A few of ESP's commands have not been analytically differentiated yet; for these, sensitivities are computed via finite-differences. A new finite-differencing technique is employed in these cases that is extremely robust, since a new "mapping" technique guarantees the correct association of points in the baseline and perturbed geometries. Unfortunately when finite differences are used, the process is less efficient than analytic derivatives, since it requires the generation of a "perturbed" configuration. It is also less accurate, since one needs to carefully select a "perturbation step" that is a balance between truncation and round-off errors.

Fig. 4 shows the sensitivity of the surface of a glider with respect to the wing's thickness.

Associative

ESP maintains a set of global and local attributes on the Faces of a configuration that are persistent through rebuilds. There are a variety of ways that attributes can be set.

Global attributes can be set to every Body in a model by defining the attribute before and Feature Tree branch. (There are no such global attributes in Fig. 1.) Also, attributes can be set on all Faces in a Body (when the Body is produced by executing a Branch in the Feature Tree). In Fig 1, lines 9a, 11a, and 15a say that all the Faces associated with the bolt's head will have an attribute named "name" whose value is the string "head".

As will be discussed below, the attribution capability of ESP is an essential capability when one generates multi-fidelity models, wherein attributes can be used to associate conceptually-similar parts in the various models. Similarly during the generation of multi-disciplinary models, attributes can be used to associate surface groups which share common loads and displacements. Lastly, attributes support the "marking" of Faces and Edges with quantities such as nominal grid spacings, material properties, boundary conditions, etc.

Extensible

As briefly mentioned above, ESP has the ability to allow users to create their own user-defined primitives (UDPs). This is especially useful when creating aerodynamic configurations, since they are seldom composed of boxes, cylinders, etc. The current version (v1.08) ships with the following UDPs:

- `bezier` — generate a Bezier Wire, Sheet, or Solid Body from a input file
- `biconvex` — generate a biconvex airfoil
- `box` — generate a (rectangular) Wire, Sheet, or Solid Body centered at the origin (with possibly-rounded corners)
- `ellipse` — generate an ellipse centered at the origin
- `freeform` — generate a freeform Wire, Sheet, or Solid Body from an input file
- `import` — read a Body out of a `.step` file
- `kulfan` — generate a Kulfan airfoil segments
- `naca` — generate a NACA 4-series airfoil or camberline
- `naca456` — generate a NACA 4-, 5-, or 6-series airfoil
- `parsec` — generate a Parsec airfoil by either specifying Sobieski's parameters or spline parameters
- `sew` — sew Faces in a `step` file into a Solid Body
- `supell` — generate a 4-quadrant super-ellipse
- `waffle` — generate a waffle by extruding a 2D group of segments

The UDPs are not directly built into ESP, but rather are dynamically loaded (when needed at run time), using a well-defined API. All the UDPs create a single primitive solid. UDPs can be written either top-down (following a procedure similar to that used in constructive solid geometry) or can be bottom-up, where the user defines Nodes, Curves, Edges, Loops, Surfaces, Shells, and Bodys. This latter bottom-up procedure is quite labor intensive to write, but once completed provides extremely rapid generation of virtually any Body. Users write a UDP in C, C++, or FORTRAN, making calls to EGADS' API, and compiles them into a dynamically loadable library.

In this way `ESP` allows for user customizable *CAD features*. Anyone who has tried to build a wing in a traditional CAD system based on specific profiles (like NACA series) can appreciate the importance of `ESP` extensibility. By this technique, not only can the tedious nature of point production and fitting be avoided, but the UDP subsystem is also parametrically driven. So in the case of NACA profiles, both the thickness and chamber can be adjusted by `.csm` scripts.

`ESP` is also extensible through the use of user-defined codes (UDCs), that can be thought of as "macros". Since these UDCs are macros, there is no limitation on the how it interacts with the stack; a UDC can pop zero or more Bodys off the stack and push zero or more Bodys back onto the stack. The current version of `ESP` ships with the following UDCs:

- `biconvex` — generate a biconvex airfoil
- `boxudc` — similar to the box UDP
- `diamond` — generate a double-diamond airfoil
- `flapz` — cut a (deflected) flap in a Body
- `gen_rot` — general rotation with two fixed points
- `popupz` — pop up a part of the configuration
- `spoilerz` — pop up a spoiler
- `duct` — generate a duct
- `fuselage` — generate a fuselage
- `strut` — generate a strut (between a duct and wing)
- `wing` — generate a wing

Deployable

`ESP` has been developed using a client-server architecture, which is similar to the architecture of many modern software systems.

`ESP`'s back-end (server) is written in C, C++, and FORTRAN, and runs on a wide variety of modern compute platforms, including various flavors of LINUX, Mac OSX, and several Windows operating systems. It is in the server where nearly all the computations are done.

`ESP`'s user-interface (client) runs in most modern web browsers, without the need for any plug-ins. The only requirements for the browser is that it support WebSockets and WebGL. At the current time, the supported browsers include FireFox, Google Chrome, and Safari.[a] The browser code, which is written in JavaScript, gives the user complete control of the graphical representation of the configuration, as well as allows the user to modify the Feature Tree (by adding, modifying, or deleting Branches) and the Design Parameters.

The entire `ESP` system is written as an open-source project, using the LGPL 2.1 license, and is distributed as source from `acdl.mit.edu/ESP`.

Explicit Design Intent

The concept of *Design Intent* it important for any design project and is where the design is defined. For an "aCAD" system, the Design Intent needs to include all multidisciplinary views or expressions. The Design Intent should be readable and is at the core of communicating what the design entails. In traditional "aCAD" and "mCAD" systems this important information only exists with the proprietary part/assembly files, frequently only in an implicit form, and is therefore opaque.

[a]Note that Internet Explorer (IE) is not supported at this time due to internal errors in IE.

As can be seen from Fig. 1, Master Models are defined in `.csm` files that are human readable (ASCII), using a stack-like language that is consistent with Feature Trees. This is where `ESP` holds the Design Intent. It also contains looping and logical decisions via "patterns" (not seen in Fig. 1). This method of defining the design makes it easy for other programs to create and/or edit any `.csm` file.

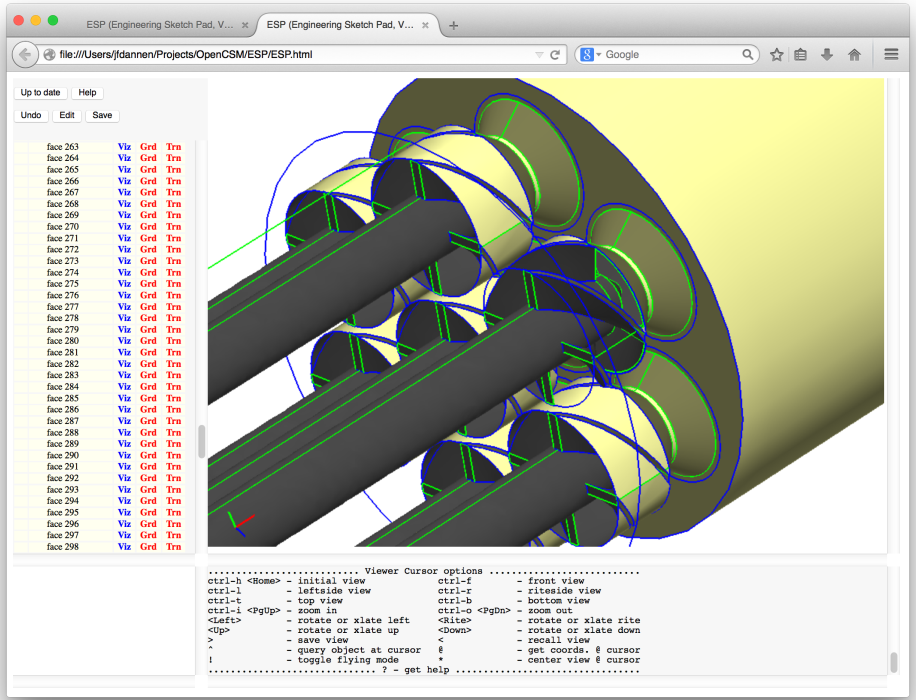
Embeddable

In addition to the `.csm` file, users can interact with the geometry subsystem (`OpenCSM`) through an Application Programming Interface (API). An application that embeds `OpenCSM` will typically load a Master Model (from a `.csm` file), interrogate and/or edit the Master Model (that is, its Feature Tree and Design Parameters), execute the Feature Tree and create a Boundary Representation (BRep), interrogate the BRep, and "set" and "get" sensitivities.

Because of the Open Source nature of the license and the modular nature of `ESP`, software components can be placed in large and complex computing environments that are in use for high-fidelity engineering analysis simulations. For example, `EGADS` can be used by CFD that does mesh adaptation, on-the-fly, to allow for the placement of points on the actual geometry. `OpenCSM` can be inserted in the gradient-based optimization loop to both drive the construction of new geometry and provide the parametric sensitivities for locations on that geometry.

III. Sample Applications

The `ESP` system is currently in use in several organizations. Figures 5 to 8 show examples of several configurations designed by or for users.

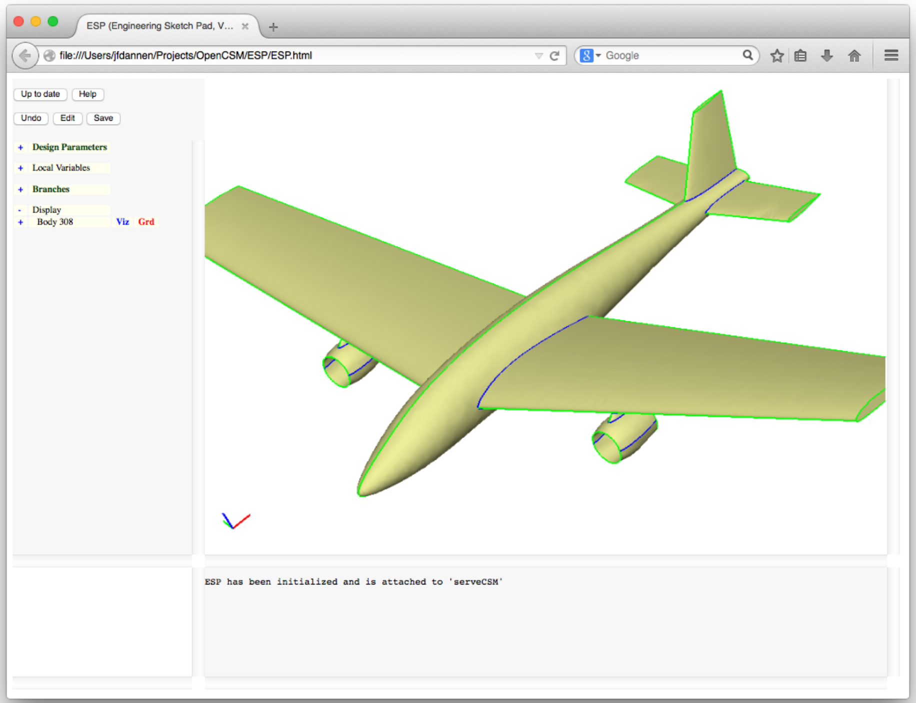

ESP (Engineering Sketch Pad, V...
ESP (Engineering Sketch Pad, V...
file:///Users/jfdannen/Projects/OpenCSM/ESP/ESP.html
Google
Up to date
Help
Undo
Edit
Save

face 263	Viz	Grd	Trn
face 264	Viz	Grd	Trn
face 265	Viz	Grd	Trn
face 266	Viz	Grd	Trn
face 267	Viz	Grd	Trn
face 268	Viz	Grd	Trn
face 269	Viz	Grd	Trn
face 270	Viz	Grd	Trn
face 271	Viz	Grd	Trn
face 272	Viz	Grd	Trn
face 273	Viz	Grd	Trn
face 274	Viz	Grd	Trn
face 275	Viz	Grd	Trn
face 276	Viz	Grd	Trn
face 277	Viz	Grd	Trn
face 278	Viz	Grd	Trn
face 279	Viz	Grd	Trn
face 280	Viz	Grd	Trn
face 281	Viz	Grd	Trn
face 282	Viz	Grd	Trn
face 283	Viz	Grd	Trn
face 284	Viz	Grd	Trn
face 285	Viz	Grd	Trn
face 286	Viz	Grd	Trn
face 287	Viz	Grd	Trn
face 288	Viz	Grd	Trn
face 289	Viz	Grd	Trn
face 290	Viz	Grd	Trn
face 291	Viz	Grd	Trn
face 292	Viz	Grd	Trn
face 293	Viz	Grd	Trn
face 294	Viz	Grd	Trn
face 295	Viz	Grd	Trn
face 296	Viz	Grd	Trn
face 297	Viz	Grd	Trn
face 298	Viz	Grd	Trn

```
........................... Viewer Cursor options ...........................
ctrl-h <Home> - initial view              ctrl-f        - front view
ctrl-l        - leftside view             ctrl-r        - riteside view
ctrl-t        - top view                  ctrl-b        - bottom view
ctrl-i <PgUp> - zoom in                   ctrl-o <PgDn> - zoom out
<Left>        - rotate or xlate left      <Rite>        - rotate or xlate rite
<Up>          - rotate or xlate up        <Down>        - rotate or xlate down
>             - save view                 <             - recall view
^             - query object at cursor    @             - get coords. @ cursor
!             - toggle flying mode        *             - center view @ cursor
.............................. ? - get help ..................................
```

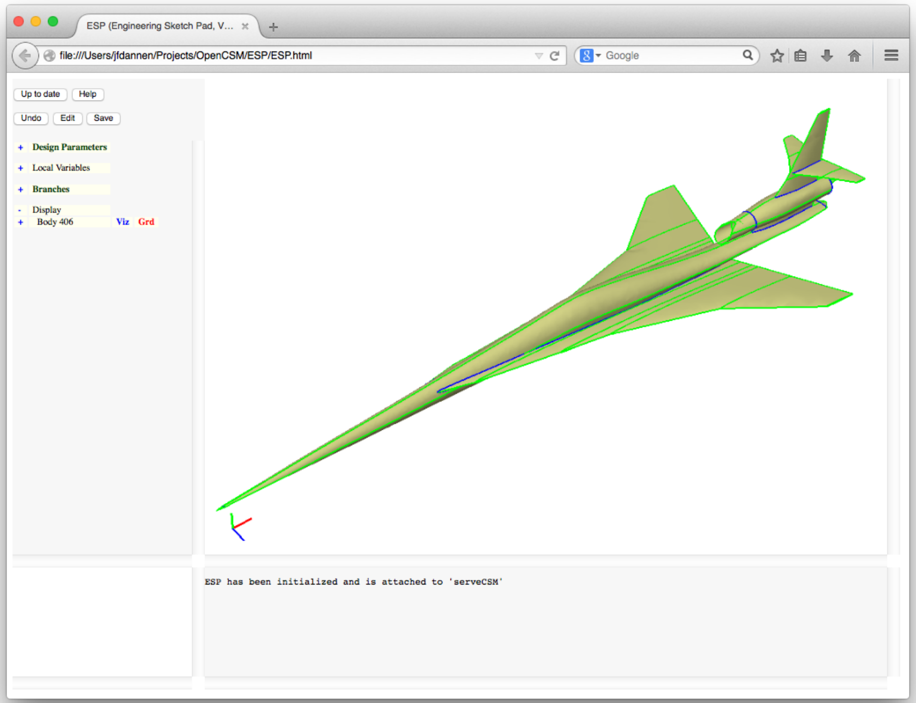
Figure 5. LDI injector configuration

Each of these configurations is quite complex, and each utilizes many of the features that `ESP` offers. Table 2 shows the complexity of these models in terms of the number of design parameters, the number of



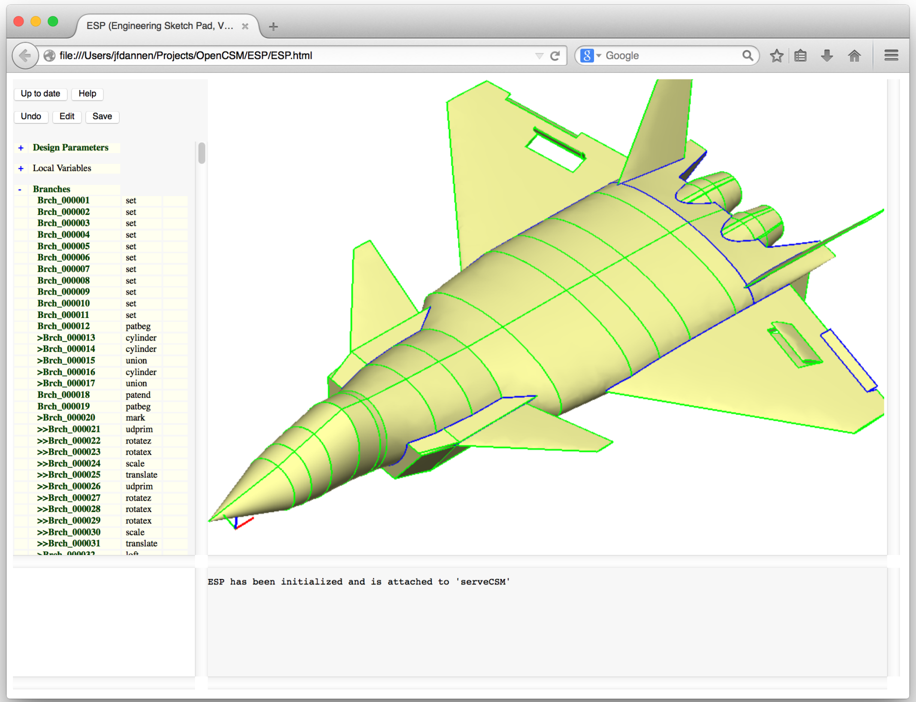
```
ESP has been initialized and is attached to 'serveCSM'
```

Figure 6. myPlane configuration, which is built from an assembly of user-defined components (UDCs)



```
ESP has been initialized and is attached to 'serveCSM'
```

Figure 7. Low Boom Flight Demonstrator


ESP (Engineering Sketch Pad, V…
file:///Users/jfdannen/Projects/OpenCSM/ESP/ESP.html
Google
Up to date
Help
Undo
Edit
Save
+ Design Parameters
+ Local Variables
- Branches

Brch_000001	set
Brch_000002	set
Brch_000003	set
Brch_000004	set
Brch_000005	set
Brch_000006	set
Brch_000007	set
Brch_000008	set
Brch_000009	set
Brch_000010	set
Brch_000011	set
Brch_000012	patbeg
>Brch_000013	cylinder
>Brch_000014	cylinder
>Brch_000015	union
>Brch_000016	cylinder
>Brch_000017	union
Brch_000018	patend
Brch_000019	patbeg
>Brch_000020	mark
>>Brch_000021	udprim
>>Brch_000022	rotatez
>>Brch_000023	rotatex
>>Brch_000024	scale
>>Brch_000025	translate
>>Brch_000026	udprim
>>Brch_000027	rotatez
>>Brch_000028	rotatex
>>Brch_000029	rotatex
>>Brch_000030	scale
>>Brch_000031	translate

```
ESP has been initialized and is attached to 'serveCSM'
```

Figure 8. J20-like model for studying control effectiveness.

branches in the feature tree, and number of operations performed during the generation of the Boundary Representation. The Table also contains statistics about the complexity of the resulting BRep in terms of the number of Nodes, Edges, and Faces that each contain.

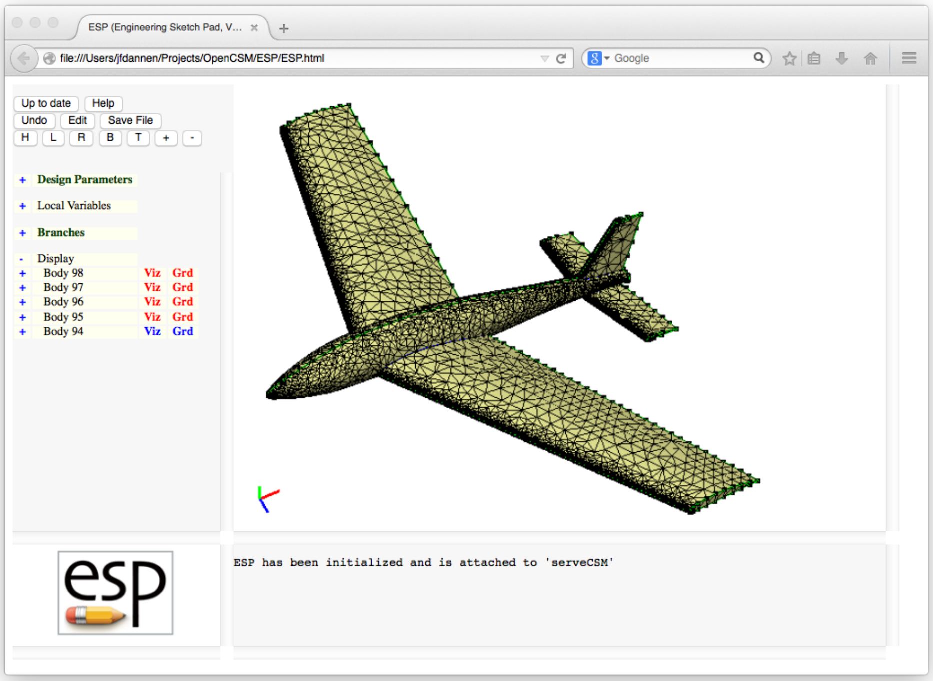
Table 2. Summary of typical aerospace configurations in ESP.

Configuration	# Design Parameters	# Feature-tree Branches	# Operations	# Nodes	# Edges	# Faces
LDI combustor	11	113	3219	615	1039	386
myPlane	63	586	615	54	89	39
Low boom A/C	234	337	406	149	268	109
J20	161	1308	1307	211	336	133

IV. Multi-Models

ESP has been designed to generate families of linked models for multi-fidelity and multi-disciplinary analysis. Figs. 9 through 11 are examples. These model were built from the same set of 33 scalar Design Parameters and six array-valued Design Parameters that define the fuselage.

Fig. 9 contains the outer mold line (OML) for the complete glider; this model is suitable for use in a field solver, such as Computational Fluid Dynamics (CFD) that could be used in the preliminary design process.



```
ESP has been initialized and is attached to 'serveCSM'
```

Figure 9. Outer mold line of glider that could be used for computational fluid dynamics.

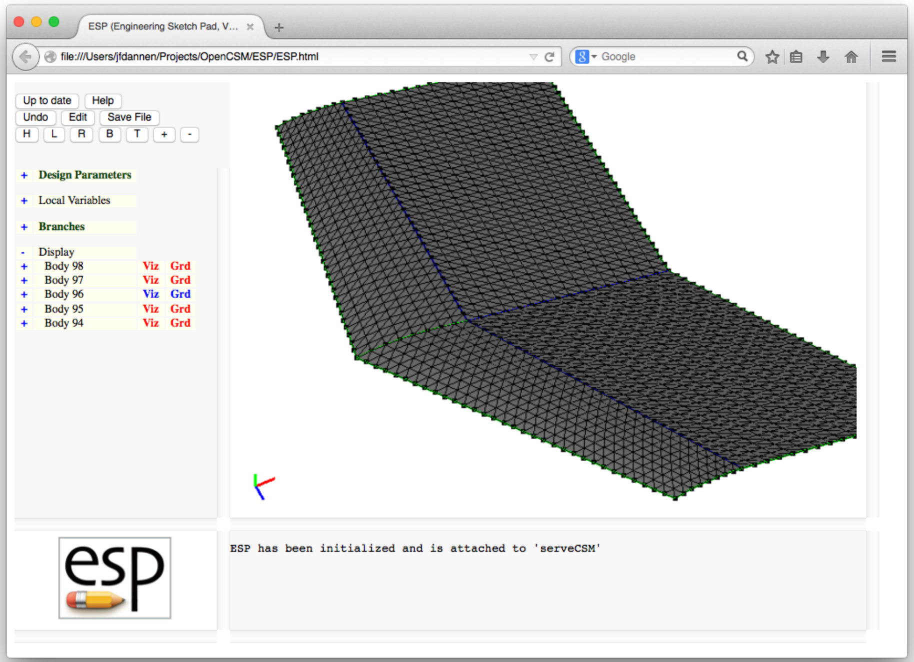
Through the same Design Parameters, it is possible to generate a mid-surface aerodynamic (MSA) model of the wing's lifting and the wake sheet that emanates from its trailing edge, and which is shown in Fig. 10. (This is an example of a non-manifold sheet Body.) Such a model could be used for a vortex-lattice method that is utilized as part of the conceptual design process. It is important to note here that both of these models (the OML and MSA) are attributed in a consistent way, such that one knows which Faces in the OML are linked to which Faces in the MSA; this is key to transferring information between the two models.

Also through the same Design Parameters, it is possible to generate a built-up element model (BEM), which is suitable for a finite element analysis (FEA) of the structure of the wing. This model, which consists of wing surface panels, spars, and ribs, is shown in Fig. 11 with its default grids of quadrilaterals. Fig. 12 shows the same BEM model with its upper surface panels removed so that the internal structure can be seen. Like the MSA, this model is not a solid Body, but instead a non-manifold collections of Faces. Again, this model is attributed consistently with the OML, such that loads could be passed from a CFD analysis on the OML to a FEA analysis on the BEM, using a conservative data transfer technique.[9]

It is worth a slight diversion to describe how the BEM was built. It started with a solid Body that represents the OML of the wing alone, as shown in Fig. 13.

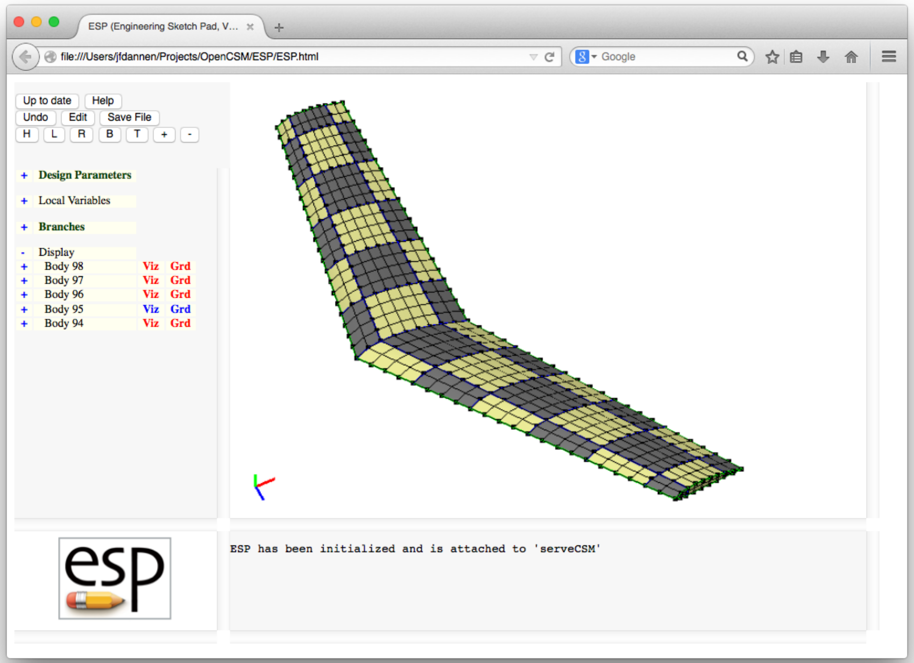
Then the non-manifold sheet Body shown in Fig. 14 was created by using the `waffle` UDP; this takes a group of crossing lines in a plane and extrudes them up to create a non-manifold group of Faces.

Once these two Bodies were available, they were intersected, which trimmed the spars and ribs to be only the parts within the OML. Then the OML was converted from a solid Body into a (manifold) sheet



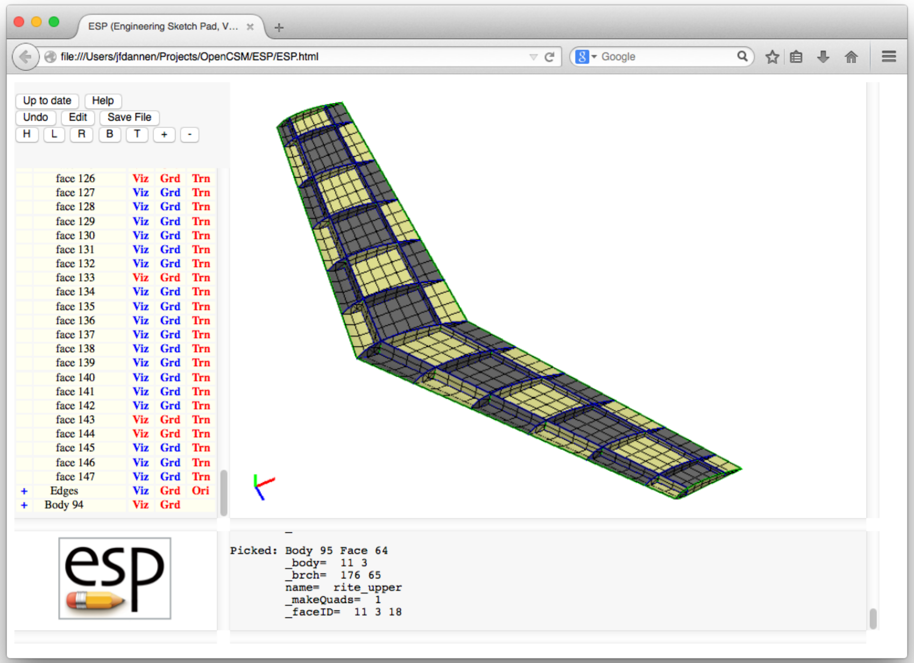
```
ESP has been initialized and is attached to 'serveCSM'
```

Figure 10. Mid-surface aerodynamic model of the wing's lifting surface and the wake sheet that emanates from its trailing edge.



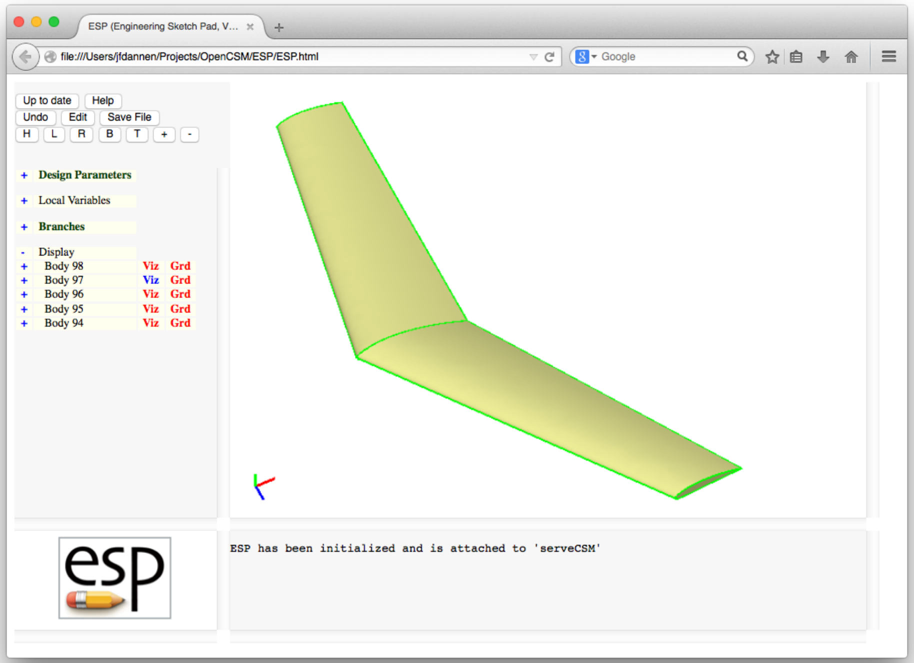
```
ESP has been initialized and is attached to 'serveCSM'
```

Figure 11. Built-up element model of the wing, with quadrilaterals and triangles associated with the surface panels, spars, and ribs.


ESP (Engineering Sketch Pad, V...
file:///Users/jfdannen/Projects/OpenCSM/ESP/ESP.html
Up to date
Help
Undo
Edit
Save File
H L R B T + -
face 126 – face 147 Viz Grd Trn
Edges Viz Grd Ori
Body 94 Viz Grd
esp

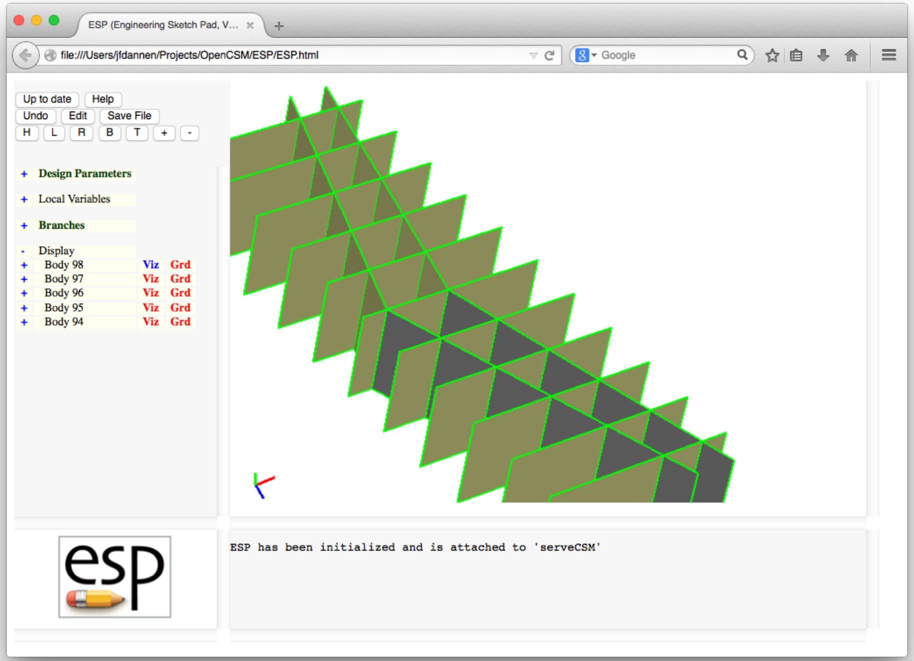
```
                -
Picked: Body 95 Face 64
        _body=  11 3
        _brch=  176 65
        name=  rite_upper
        _makeQuads=  1
        _faceID=  11 3 18
```

Figure 12. Built-up element model of the wing, with quadrilaterals and triangles associated with the surface panels, spars, and ribs. The upper surface has been removed so that the inside can be seen.



```
ESP has been initialized and is attached to 'serveCSM'
```

Figure 13. Outer mold line of wing.



```
ESP has been initialized and is attached to 'serveCSM'
```

Figure 14. Waffle generated by a UDP that contains untrimmed spars and ribs.

Body that consisted of the OML's Faces. The waffle was then subtracted from the outer sheet Body, which caused the outer sheet Body to be scribed by the faces of the waffle; this resulted in the outer skin panels of the wing. Finally the two parts were unioned, resulting in the BEM.

V. Summary

The Engineering Sketch Pad (ESP) has been described and its distinguishing features were identified. These included:

- solid models — models are guaranteed to be realizable, watertight solids (unless purposely built as a sheet or wire Body).
- feature-based — models are built using the constructive solid modeling paradigm; the features can easily be suppressed to make simpler models.
- parametric — a set of externally-controllable Design Parameters are used to drive all aspects of model generation; in this way, a single model can be used in a design setting to create Bodys of different sizes and shapes.
- differentiated — sensitivities of all aspects of a Body (and a grid on it) with respect to the Design Parameters can be generated analytically for many configurations without regeneration. For cases where this is not possible, a robust finite-difference technique is automatically invoked.
- associative — global and local attributes, which can be applied to various parts of a Body, are persistent through rebuilds. This capability is key to the linking of multi-models.
- extensible — users can write user-defined primitives (UDPs) for non-standard shapes (such as airfoil profiles), which can be loaded dynamically by ESP. This makes ESP particularly well suited to

aerodynamic configurations, which require great control over the form of the craft.

- deployable — the client/server architecture is freely-downloadable as source that is licenced under LGPL 2.1. The server runs on nearly all variants of LINUX, OSX, and Windows, and the client uses most modern browsers, such as FireFox, Google Chrome, and Safari.
- explicit Design Intent — the design is fully described using `ESP`'s input `.csm` files, which are ASCII, human readable and can easily be manipulated with any text editor or other program.
- embeddable — the Application Programming Interfaces (APIs) allows `ESP` to be embedded into other software environments of various levels of complexity.

The above makes `ESP` ideally suited for the generation of multi-disciplinary and multi-fidelity models. The key to the use of these multi-models is the ability to link the models together. In other words, it is important to know which surface in model A is associated with which surface in model B, especially if one wants to transfer load and/or displacements amongst the models. Fortunately, `ESP` contains a robust set of attributions that are carried throughout the regeneration process.

Acknowledgements

This work was funded by two projects. The first is the NASA Cooperative Agreement NNX11AI66A: Topic 2.1 - MDAO Open-source Engineering Framework; Chris Heath was the Technical Monitor. The second is the CAPS project, which was funded under AFRL Contract FA8050-14-C-2472: "CAPS: Computational Aircraft Prototype Syntheses"; Ed Alyanak is the Technical Monitor.

References

[1]Coons, S.A., and Mann, R.W., "Computer-Aided Design Related to the Engineering Design Process", Technical Memorandum 8436-TM-5, Electronic Systems Laboratory, MIT, October 1960.

[2]Waguespack, C., "Mastering Autodesk Inventor 2015 and Autodesk Inventor LT 2015", John Wiley & Sons, 2014, pp 19.

[3]Snepp, D.K. and Pomeroy, R.C., "A Geometry System for Aerodynamic Design", AIAA-87-2902 January 1987.

[4]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.

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

[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]http://www.opencascade.org

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

[9]Dannenhoffer, J.F. and Haimes, R., "Conservative Fitting for Multi-Disciplinary Analysis", AIAA-2014-0294, January 2014.