An Overview of the Engineering Sketch Pad

John F. Dannenhoffer, III

jfdannen@syr.edu

Syracuse University

AIAA SciTech 2024
January 10, 2024

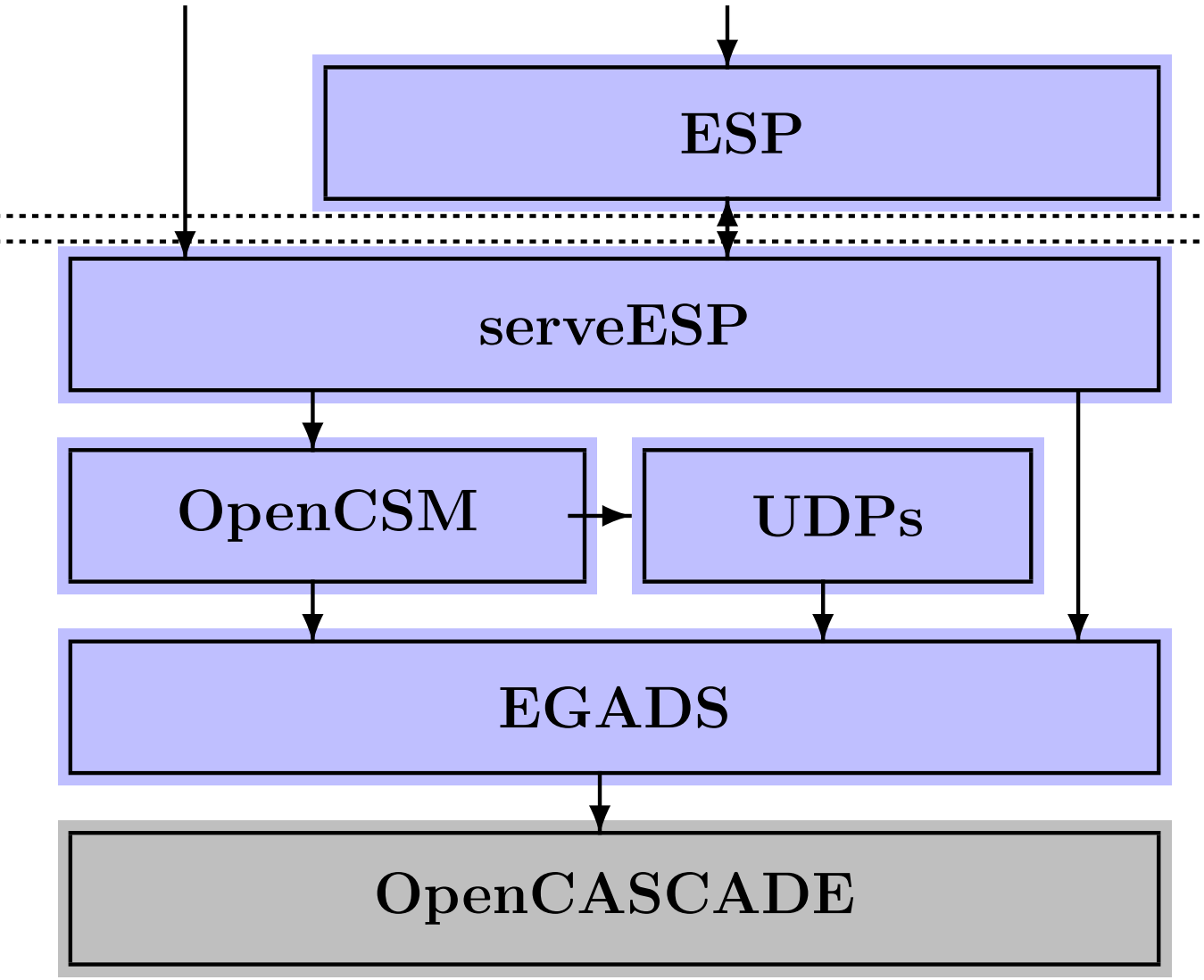
Overview

- Background
- ESP's architecture
- Creating geometry in ESP
- Example of a `.csm` script
- Geometric and tessellation sensitivities
- Using ESP for geographically-dispersed teams
- Using Python to automate tasks
- ESP extensions
- Availability and training

Engineering Sketch Pad (ESP)

- ESP is a geometry creation and manipulation system designed specifically to support the analysis and design of aerospace vehicles
  - feature-based solid modeler
  - parametric
  - associative via rich user-defined attribution
  - differentiated
  - scripted, with loop, logic, and error recovery
  - extensible via UDPs, UDFs, and UDCs
  - executes on Windows, Linux, and MacOS
  - user interaction via all modern browsers
  - freely available from `acdl.mit.edu/ESP`

ESP Architecture



user interactions

system driver
create graphic objects

design parameters
feature tree
user-defined primitives
sensitivities
top-down & bottom-up
persistent attribution
watertight tessellation

geometry primitives
boundary representations

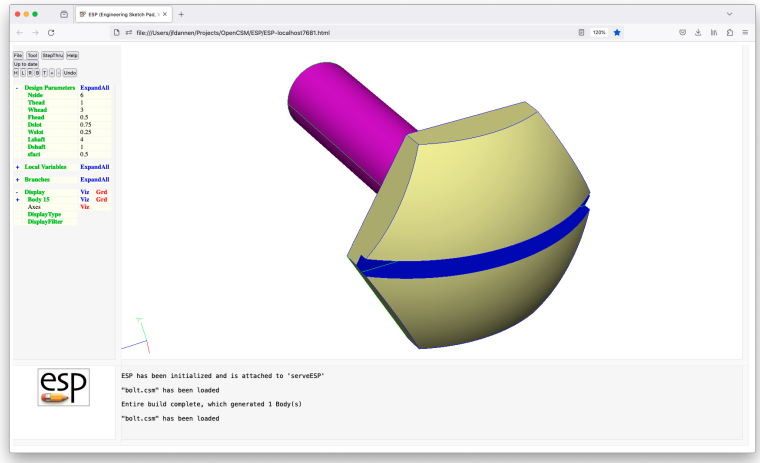
Creating Geometry in ESP

- Define design and configuration parameters
- Generate the base Body(s)
  - primitives: box, sphere, cylinder, cone, torus, import, user-defined primitive (UDP)
  - grown: extrude, revolve, rule, blend, sweep
- Apply attributes
  - user-defined name
  - values can be (matrix/array of) numbers or strings
- Transform the Bodys
  - translate, rotate, scale, mirror
- Combine the Bodys
  - union, join, intersect, subtract

Note: all this can be done via the browser or by writing a `.csm` script.

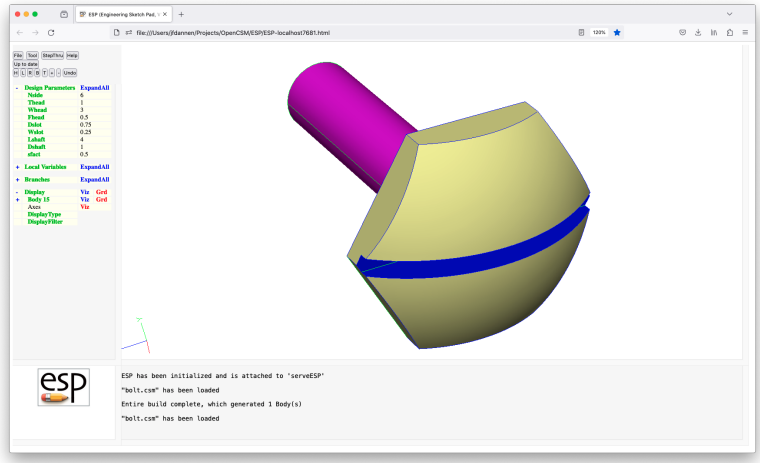
Example of a `.csm` script — 1

```
# bolt example
# design parameters
CFGPMTR   Nside     6      # number of sides

DESPMTR   Thead     1.00   # thickness of head
DESPMTR   Whead     3.00   # width     of head
DESPMTR   Fhead     0.50   # fraction  of head that is flat

DESPMTR   Dslot     0.75   # depth of slot
DESPMTR   Wslot     0.25   # width of slot

DESPMTR   Lshaft    4.00   # length   of shaft
DESPMTR   Dshaft    1.00   # diameter of shaft

DESPMTR   sfact     0.50   # overall scale factor

# make sure the number of sides is even
IFTHEN    mod(Nside,2) NE 0
   MESSAGE the_number_of_sodes_must_be_even
   THROW   -999
ENDIF

# head
BOX       0      -Whead/2  -Whead  Thead  Whead  2*Whead
PATBEG    iside  Nside/2-1
    BOX       0  -Whead/2  -Whead  Thead  Whead  2*Whead
    ATTRIBUTE tagComp  $head
    ROTATEX   360*iside/Nside
    INTERSECT
PATEND
```



Example of a `.csm` script — 2

```
SET       Rhead  "(Whead^2/4+(1-Fhead)^2*Thead^2) \
                  /(2*Thead*(1-Fhead))"

SPHERE    0              0   0    Rhead
ATTRIBUTE tagComp  $head
TRANSLATE Thead-Rhead    0   0
INTERSECT

# slot
IFTHEN    Dslot GT 0   AND   Wslot GT 0
   BOX       Thead-Dslot  -Wslot/2  -Whead  2*Thead  Wslot  2*Whead
   ATTRIBUTE tagComp  $slot
   ATTRIBUTE _color   $blue
   SUBTRACT
ENDIF

# shaft
CYLINDER  -Lshaft   0   0   0   0   0   Dshaft/2
ATTRIBUTE tagComp  $shaft
ATTRIBUTE _color   $magenta
UNION

SCALE     sfact

END
```

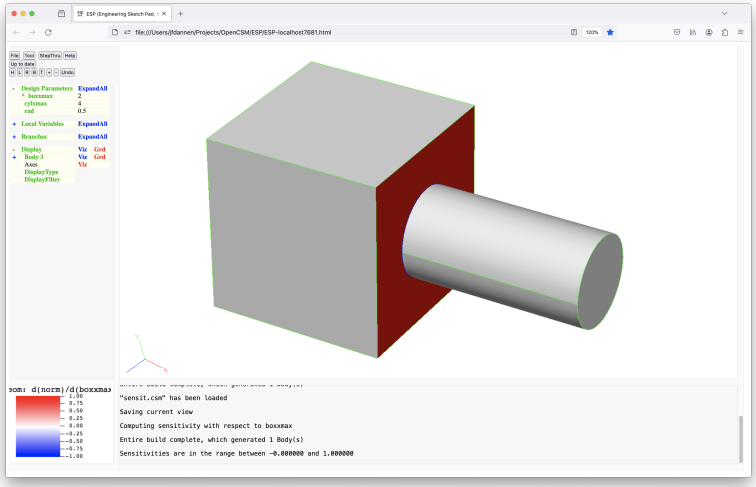


Geometric and Tessellation Sensitivities — 1

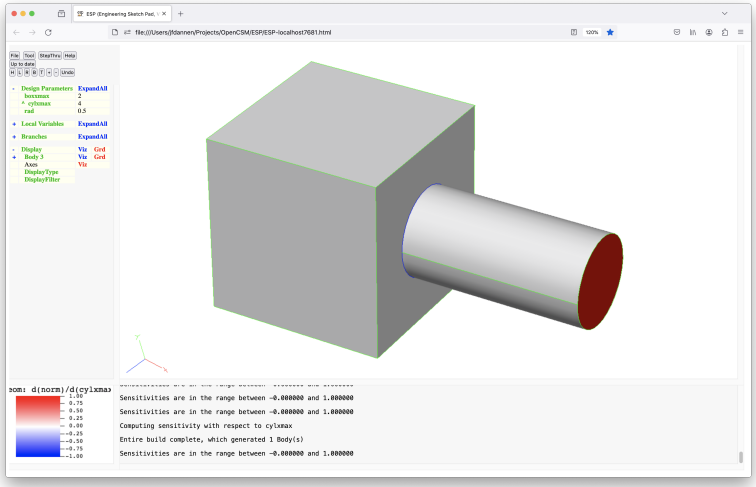
- ESP computes two type of sensitivities:
  - **Geometric sensitivities** — describe the motion normal to a surface (Face), perpendicular to an curve (Edge), or of a Node when the design parameter is changed
    - these are "exact"
    - these are often discontinuous at Edges
  - **Tessellation sensitivities** — describe the motion that a grid point on a surface *might* have taken when the design parameter is changed
    - these are approximated by interpolating the sensitivities of the trimming Edges (for Faces) or trimming Nodes (for Edges)
    - these are continuous at Edges (and Nodes)
- Sensitivities are computed:
  - analytically for most operations — these are "exact"
  - via finite differences — these are approximate because of truncation and round-off errors

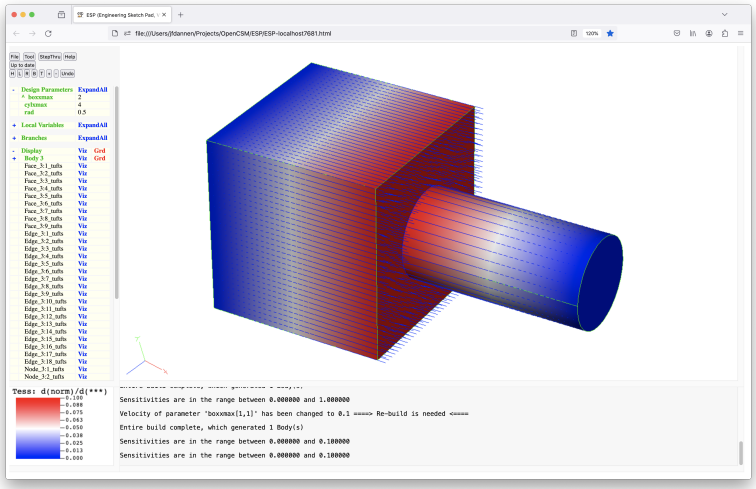
Geometric and Tessellation Sensitivities — 2

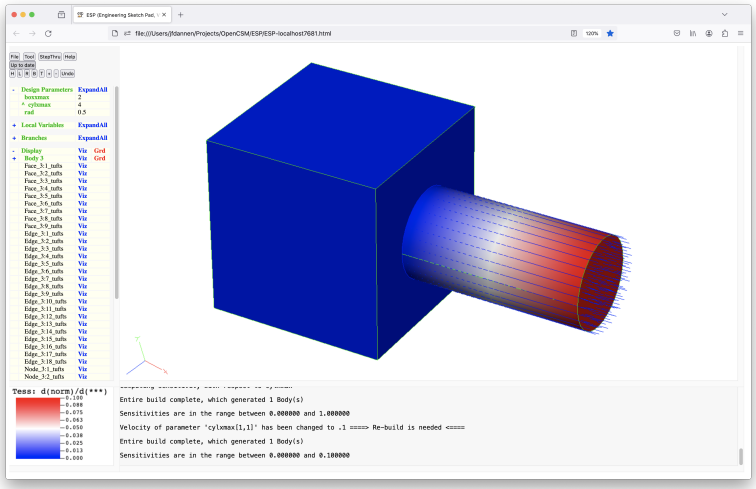
`box:xmax`



`cyl:xmax`







ESP's Collaboration Environment — 1

- Collaboration is two or more people working simultaneously to achieve a single result
- A "best-practice" in software engineering (called "pair-programming")
  - one person at the keyboard - focuses on tactical (detailed) decisions
  - other person(s) looking over the shoulder - focus on strategic (higher-level) decisions
  - studies show that
    - cumulative labor for pair-programming is only about 40% more than working alone
    - BUT — less tendency to get distracted and take shortcuts
    - AND — product has far fewer defects
  - frequent switching of roles is essential for success
- Also can be applied when two workers cannot be co-located

ESP's Collaboration Environment — 2

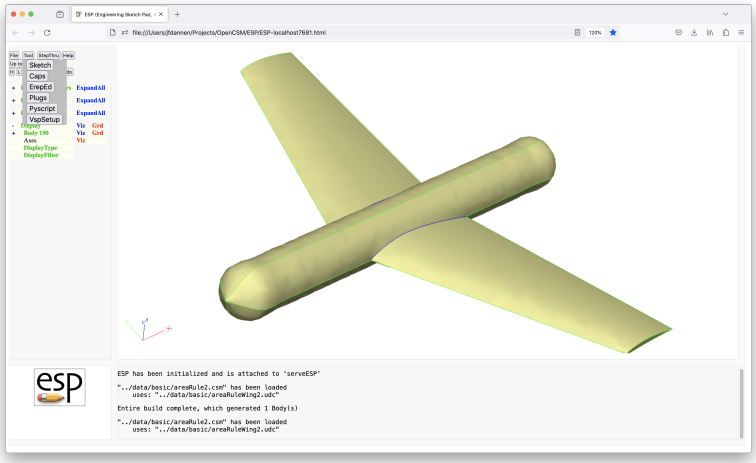
- All users see the same model (as it is being built)
- One user "has the ball" and is in complete control of:
  - model being generated
  - his/her own screen
  - passing the "ball" to another user
- Other users can:
  - synchronize his/her display to the user with the ball, or
  - have control of own screen, but not changes to the model
  - request (and accept) the "ball" at any time
- Case studies in ESP has shown that collaboration feature retains all the benefits of "pair programming"
  - shared ownership of model
  - tendency to take fewer shortcuts
  - low error rate

Using Python to Automate Tasks — 1

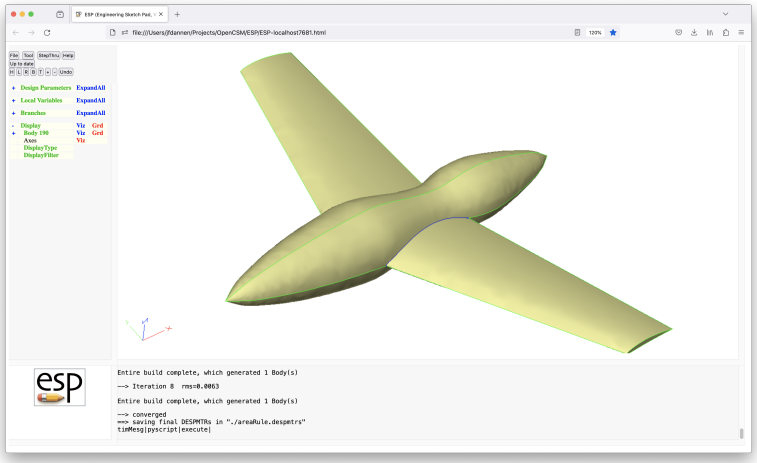
- `ESP` has an embedded Python interpreter to automate tasks
  - `pyOCSM` gives access to all `OpenCSM` operations
  - `pyEGADS` gives access to all `EGADS` operations
- Example: modify the radius of the fuselage to match the Sears-Haack area rule
  1. read the definition of the aircraft (with a guessed distribution of fuselage radii)
  2. generate the configuration and compute the cross-sectional area distribution
  3. compare the area distribution with the Sears-Haack distribution
  4. adjust the radii and repeat to step 2

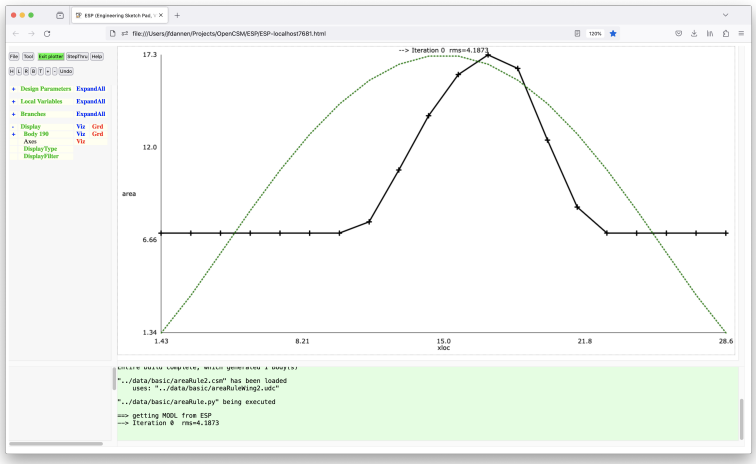
Using Python to Automate Tasks — 2

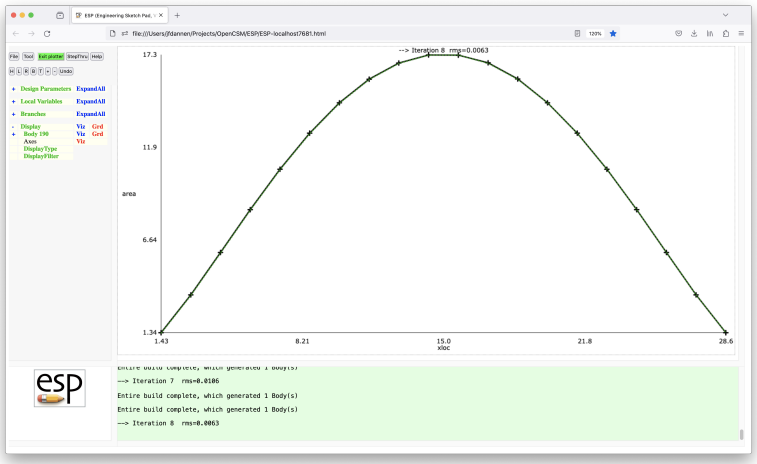
Original guess



After 8 iterations





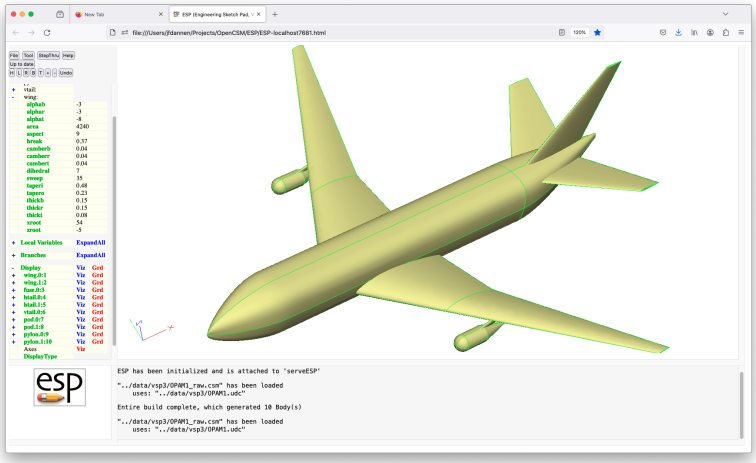


ESP Extensions

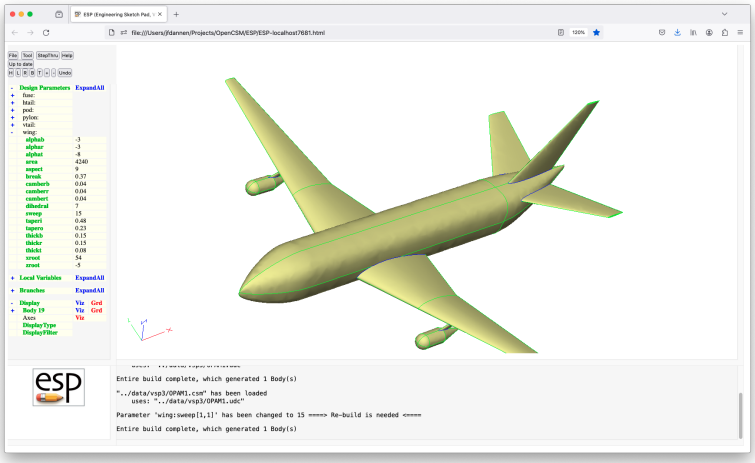
- **Sketch** — make a constrained sketch in two dimensions
- **Caps** — access to the CAPS system, which offers coupling to a variety of mesh generators, flow solvers, and structural analyses
- **ErepEd** — logically combine Faces into a quilt (called an EFace - or Effective Face)
- **Plugs** — modifies the design parameters in a user-supplied model such that it fits the model, in a least-squares sense, to a cloud of points
- **Plotter** — display multiple line plots, with control over line and symbol types and colors
- **Slugs** — build up a non-parameterized configuration by least-square fitting Faces (and Edges) to a cloud of points
- **Pyscript** — the Python interpreter
- **VspSetup** — import an `OpenVSP` model into `ESP`.

OpenVSP Example

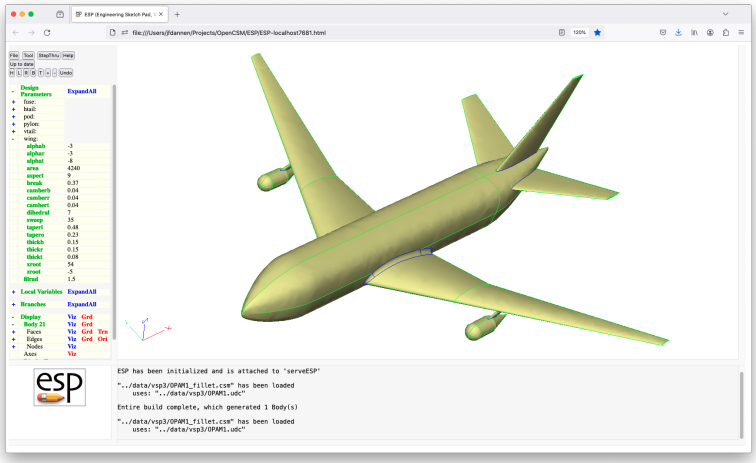
Original model (imported from OpenVSP)



wing : sweep = 15°



Fillet added



Availability and Training

- ESP is an open-source project (using the LGPL 2.1 license) that is distributed as source, and is available from `https://acdl.mit.edu/ESP`.
- That website contains:
  - the latest released software, compiled for Windows, Linux, MacOS (both Intel and ARM);
  - the full source for the latest release;
  - a list of most of the ESP-related publications;
  - copies of the slides and examples for the latest training, as well as recordings of the training; and
  - an archive all previous versions (source only) as well as a beta release of the latest stable snapshot of the repository.
- Training sessions are available to everyone and are held roughly once a year
  - next session will be early summer 2024
  - send a note to `jfdannen@syr.edu` to get on the mailing list